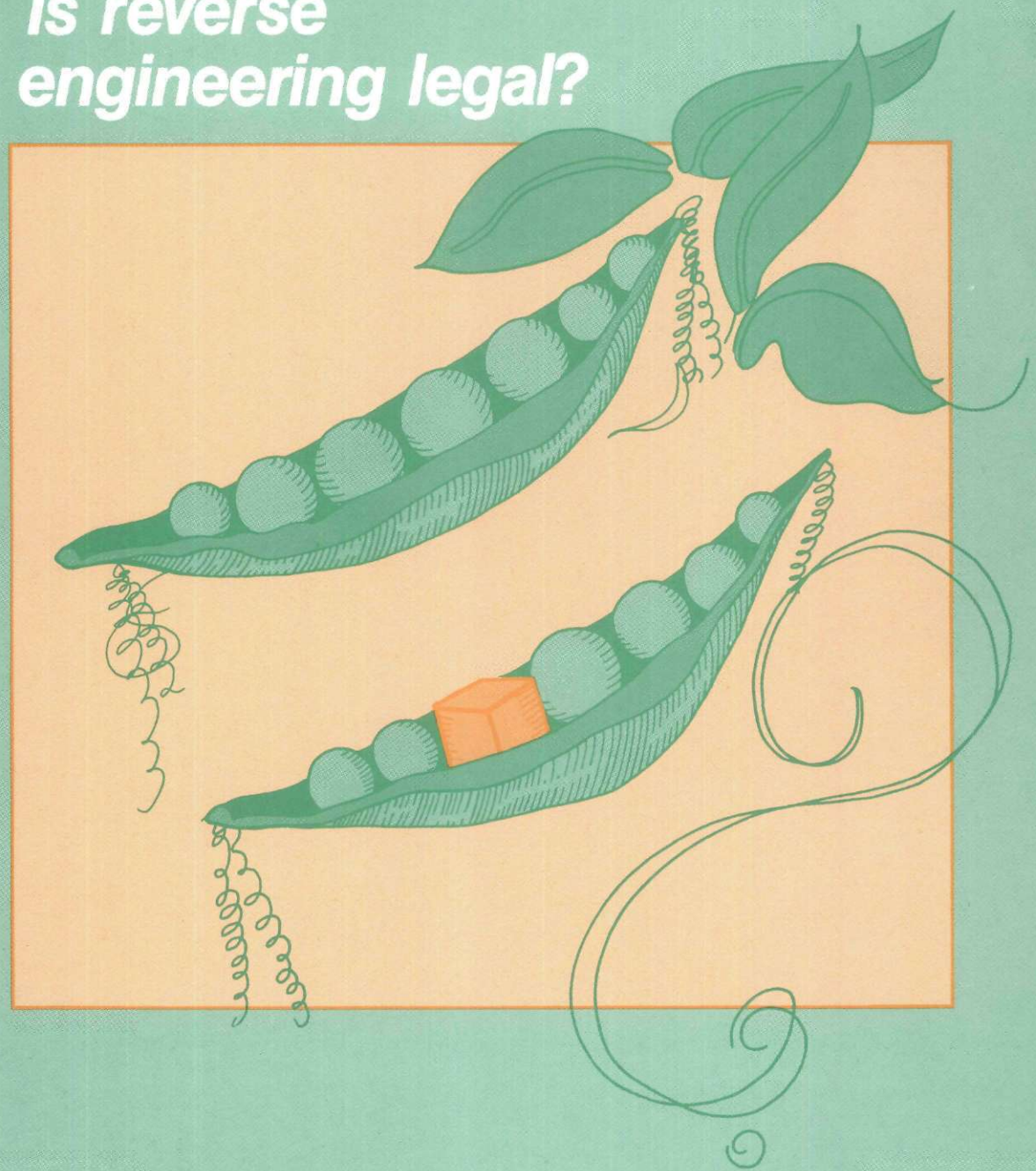
$4.00

# programmer's journal

*The Resource Journal for IBM PC Programmers*

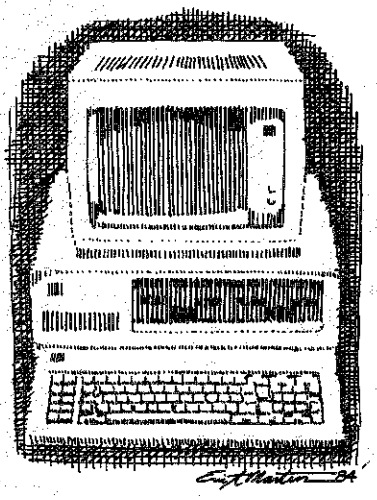
## Is reverse engineering legal?



Volume 2 Number 4

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

# PROGRAMMER'S JOURNAL



## CONTENTS


HOUSEKEEPING . . . . . 2

LETTERS . . . . . 3

PJ FORUM: BUSTED BASIC AND MASS-MARKET UNIX from Dtack Grounded . . . . . 4
*All the news that's fit to print*

PJ PROFILE: WRITING A CUSTOM ROM BIOS FOR PC-DOS/MS-DOS MACHINES . . . . . 9
*An interview with Neil Colvin, President of Phoenix Software*

REDIRECTIONS: RISC, TRANSPUTERS, & MASS STORAGE by Frank D. Greco . . . . . 13
*The Unix and PC saga continues*

THE FUTURE OF DISK PROTECTION by Richard Larratt . . . . . 17
*Who should be protecting what from whom*

ASM: 8088 ASSEMBLY MACROS by Ron Burk . . . . . 21
*Several effective and time-saving MASM Macros*

BASIC: RANDOM ACCESS FILES by Ronald Moak . . . . . 25
*What you need to know first*

C: C UTILITY ROUTINES by Warren Hillman . . . . . 29
*An example of C routines you can purchase*

TOOLS: USING THE PMATE MACRO LANGUAGE by Robert E. Brown . . . . . 30
*Taming Pmate's macro facility*

dBASE II: KMAN, dBASE II, & R:BASE by Dr. John W. Gowens . . . . . 38
*Which database manager should you really buy?*

PASCAL: ES/P, A SYNTAX-CHECKING PASCAL EDITOR by Jim Gore . . . . . 45
*Write your code and leave the driving to ES/P*

MISC. . . . . 47


ISSN#0747-5861



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

# HOUSEKEEPING

HOUSEKEEPING

You reset button fans have hit the jackpot. Minutes before I sat down to write this brief intro to PJ, the phone rang and Lou LaPointe gave me the good news. He has been using a reset button of his own making for over a year now to do assembly level development work. He has decided, urged on by some venture capital, to share his gem with the rest of us. For $29.95 no less. What you see is what you get. A reset button that does a Ctrl-Alt-Del when your keyboard is locked up, rather than being forced to turn your computer on and off. Just hit the button like on an Apple computer. Don't let your chips go through the wringer. For the full scoop, get in touch with:

*Lou LaPointe*
*1076 Hatch*
*St. Paul, MN 55103*
*(612) 489-5892*

Sybex, the computer book publisher, has just released a new hardback called "Confessions of an Infomaniac." Sound familiar? It is written by a woman who explores the on-line BBS's around the country, especially CompuServe and the Source. What is so intriguing, other than the title, is the woman's philosophy. She wants to meet an intelligent partner. You know, one that is talented, high tech, makes a good salary, and loves computers. Obviously she knows where to look. The on-line services have 97% male membership, the average user's salary is over $48,000, and professional software people are definitely intelligent. I only wish they would set up a BBS that women loved to use so I could search through their membership list. Now THAT is effective use of computer time.

-- Greg
□


PROGRAMMER'S JOURNAL

Published by Avant-Garde Publishing Corp.

Business Office:
Avant-Garde Publishing Corp.
1907 Garden Ave.
Eugene, OR 97403
(503) 345-3043

Editorial Office:
2765 Potter St.
Eugene, OR 97405
(503) 687-0398

Subscriptions and inquiries:
P.O. Box 30160, Eugene, OR 97403
(503) 345-3043

Programmer's Journal. Volume 2, Number 4.


Copyright © 1984 by Avant-Garde Publishing Corp.


Published bi-monthly.

Postmaster send address changes to:
Programmer's Journal
P.O. Box 30160
Eugene, OR 97403
(503) 345-3043

Publisher: Mary Carol Smith
Associate Publisher: Elizabeth Oakley
Editor: Greg Estes
Copy Editor: Joan Aschim
Production Manager: Christine Baldwin
Production Assistants: Marilyn Reaves, Jennifer Mullock
Cover Design: Pamela Shute
Illustrations: Julia King, Eric Martin

Contributing Writers:
Ron Moak
Frank Greco
Richard Larratt
Jim Gore
FNE
Robert E. Brown
Ron Burk
John W. Gowens
Warren Hillman

Subscriptions: Bi-monthly rates are $24 U.S.
($30 in Canada, and $40 elsewhere)



IBM and Personal Computer are trademarks of International Business Machines, Armonk, New York; dBASE II is a trademark of Ashton-Tate.





PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

# LETTERS

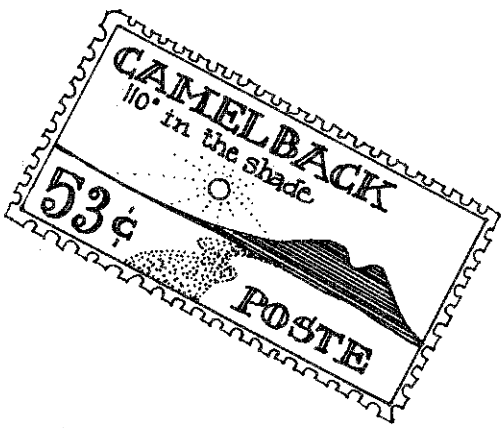
HIDDEN TREASURE

While browsing around in IBM's RECOVER utility with an un-assembler, I found an unexpected response to certain input. Namely, type in "RECOVER B:" for example, and at the prompt "Press any key..." enter a Ctrl-W. The usually straight-laced software responds with...

[Editor's note: the actual response has been deleted. YOU WILL HAVE TO TRY IT FOR YOURSELF, and I assure you it is quite interesting.]

Have fun.

*Grey Staples*
*Camelback Systems*
*Scottsdale, AZ*



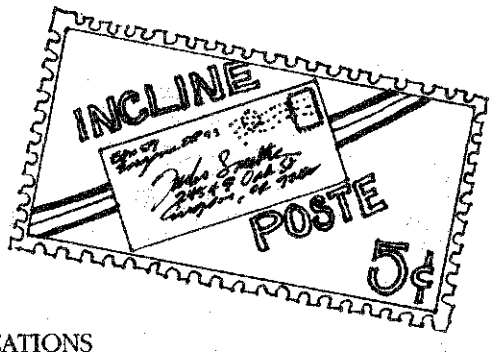

BETA JUNKIES

We're looking for a few IBM PC junkies to beta-test our new office-home checkbook system. For $30, testers get the advanced version of Money Street. They get a ready-to-use program complete with documentation. When released, it will retail for $100. We want testers to bash it, smash it, and try to crash it. Testers hunt for bugs, critique documentation, suggest program improvements, and generally help us fine-tune the program.

In return, the beta-testers get the final version of Money Street free, a tester's newsletter, and the right to purchase future software at a discount. Besides getting a $100 program for $30, testers get a peek at the nutty side of software development. Just to get their blood pumping, we hid a few bugs in the program. The bug-catchers will get some off-beat prizes and a chance to become semi-famous in computerdom.

Testers can get their copy by sending $30 to:

Bob Payne
Computer Tax Service
Box 7915-R
Incline Village, NV 89450
(702) 831-4300



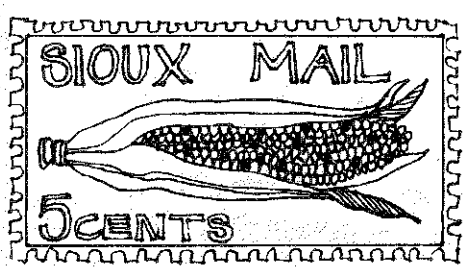

7-UP QUALIFICATIONS

Did you send in your "qualification" card for a subscription to PC-WEEK? Of course you did. Did they ignore your request? Of course they did. Are you wondering why those cards never seem to result in a free subscription? Of course you are. It's high time to "get with the program" and start receiving tons of free subscriptions. You just need to present a better corporate profile, that's all. The publishing companies are not going to send free magazines to just anybody. AND YOU ARE NOT JUST ANYBODY. RIGHT!

Are you a "manufacturer of computer-related equipment?" Of course you are. Are you the "VP of Finance/Comptroller?" Most certainly. Do you plan to buy a LARGE number of 3270 PC's in the next six months? Definitely—at least ten XT-370's for a start. Are you just a user of computers or do you approve the purchase of them? (What kind of question is that?) Of course you check off the final approval box on all request chits. In fact, you are the sole recommending source for every piece of computer equipment purchased by your multinational corporation. Who do they think is filling out this subscription qualification card anyway, some lowly programmer?

*Warren Hillman*
*Sioux Ware Systems*
*Drain, OR 55555*

□






PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

# PJ FORUM

*EXCERPTS FROM DTACK GROUNDED*
*THE 68000/16081 SYSTEMS JOURNAL, Issue No. 30*
*by FNE, Copyright © 1984 by Digital Acoustics, Inc.*

## BUSTED BASIC

Most of you are aware that Microsoft's Bill Gates has announced that Microsoft is developing an unbusted BASIC for Macintosh, one that will be (is predicted to be) ten times faster than Microsoft's BASIC for the IBM PC (which is written largely in 8080 assembly). But the Microsoft BASIC that you can buy NOW is the carefully busted 68000 BASIC that Microsoft wrote to protect Big Blue back when they were still in bed together. John M. has just sent us the figures for Terry Peterson's benchmark: 621 seconds! Remember, the Apple II runs the benchmark in 488 seconds under Applesoft. And the "accuracy" is only 4.19E-11 or ten times less accurate than the HALGOL results, and over 33 times slower! [Editor's note: HALGOL is a language written for, and used on, Digital Acoustic's 68000 board for the Apple.]

Did you get that? THIRTY-THREE TIMES SLOWER? Look, folks: it is absolutely impossible to make a 68000 run that slowly unless one does so on purpose. Those of you who have asserted that Microsoft did NOT deliberately bust its non-IBM BASICs (the ones written after IBM's entry into the personal computer marketplace) on purpose haven't a leg to stand on.

However, there is another firm which is busting its BASIC (and other languages) by accident—they don't know what they are doing! We're referring to a firm known as Digital Research, which obviously needs to do some digital research. The following quotation is from Electronics magazine, February 23, 1984, p. 100:

"The key to Digital Research's technical strategy is to achieve portability of languages and operating systems by writing them all in C. The language compiler itself, not the application program, is then portable from microprocessor to microprocessor, and applications can be ported from one system to another just by being recompiled.

"'We have completely abandoned assembly language,' notes DRI systems-software director Beebe. 'It makes you write every new generation of products from scratch.' Although assembly code tends (TENDS? YE GODS! — FNE) to be tighter and faster than high-level languages, cheap memory and high-performance processors make these advantages less important."

Isn't this where we came in a couple of years back? Where have those DRI guys BEEN lately? Are they blind and deaf or just extraordinarily dumb? How many BASICs written in C do they think they are going to sell now that Microsoft has decided not to protect IBM's interests? How many spreadsheets written in C are going to sell now that Mitch Kapor has become wealthy, proving that businessmen like software that runs fast?

Many of you may not know that DRI has been trying to sell an 8080 BASIC written in C in recent months. Yes, that BASIC is just as slooow as it implies, and Microsoft's competing 8080 BASIC is written in 8080 assembly, you will recall. So how are sales of that DRI busted-BASIC doing? Well, you hadn't heard of it until we told you about it, right?

Perhaps DRI now plans to write their C in PASCAL to gain the approval of the computer science theocracy, then write their Pascal in COBOL to please the data processing managers, and finallly write their COBOL in ADA to meet mil specs. After all, " . . . cheap memory and high performance makes (efficiency) less important." Mama mia!

## THE ONSLAUGHT COMETH! (YAWN)

AT&T, which is no longer restricted by law, is planning to enter the computer marketplace, including the personal computer marketplace. Considering the resources that AT&T commands, we are surprised how little one reads about that elsewhere in the personal computer press. Let us explain why you have not read much about that here, either:

AT&T developed a lot of software (C and Unix) while its profits were protected from competitive pressures — it was a utility whose profitability was assured by a federal regulatory agency, remember. Having been born in a noncompetitive environment, C and Unix were then nurtured in yet another noncompetitive environment — the universities. It should therefore come as no surprise that C and Unix comprise an absolutely perfect software system for noncompetitive environments.

At UCI, which happens to be located near us, 20 seconds is the usual and accustomed response time for a student sitting at a terminal ("on a good day," our other HALGOL programmer, a recent UCI computer science graduate, tells us). In such an environment, Unix is a perfectly normal operating system. We can't see a 20-second response time being tolerated in a competitive environment.

A Motorola systems type recently pointed out to us the significant difference between Motorola's VersaDOS and Unix: if three programmers are using the same compiler, only one copy of the compiler is in the system (not true of Unix). If these three programmers have each requested enough



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

memory so that none is left, a fourth programmer attempting to log onto the system is told, in effect, "The system is fully loaded — go away!" Unix just does more thrashing back and forth to disk, and the response time gets longer and longer. VersaDOS does almost no thrashing, while Unix is ALWAYS thrashing back and forth. Everything in Unix is a file, the files talk to other files, and they are all on the disk!

We don't like multi-user systems (to put it mildly), but it seems to us that VersaDOS is designed for a competitive environment. Unix isn't, due to the environment where it was born and the other environment where it was nurtured.

The AT&T folks likely see themselves as a fierce, powerful giant about to be unleashed to scourge the landscape. We see AT&T as a clumsy and unworldly oaf who is likely to be unmanned if he tries to venture out from beneath Ma Bell's skirt. Trying to tackle the MASS, competitive marketplace with C and Unix — and that is AT&T's plan — is like the Czech cavalry trying to counter German tanks and Stukas with swords and lances.

(Did you know that Commodore is now making 400,000 C-64s a month? And not keeping up with demand? That if the installed base of C-64s was over two million as of December 1, 1983 then the installed base of C-64s will be OVER FOUR MILLION by May 1, 1984? Did you know that almost all of the software sold to support the C-64 is written in untransportable assembly? With that installed base, do you think anybody CARES?)

Even the Apple Computer Company has finally figured out that you win with assembly and lose — and die — with high-level. (Place fresh flowers occasionally on LISA I's grave.) When an unbusted 68000 BASIC for the Macintosh arrives and joins their assembly-based operating system, Macintosh is going to become VERY popular, as we have been predicting for some time now.

And DRI has just "completely abandoned assembly language." Gee, what a shrewd move on DRI's part!

## MORE C-BASED SOFTWARE

The current Multiplan and BASIC which are available from Microsoft for the Macintosh were both written a while back for the 68000 at a time when Microsoft was protecting its IBM connection. Therefore, both are lovingly busted in C and hence horribly slow. Terry Peterson's benchmark, the one which runs in 18.6 seconds in HALGOL, oozes in at 621 sec using Macintosh's busted-BASIC, or 33.4 times slower! We have just spoken with two folks who got Mac's in the past week; one had had his Mac for a whole three hours. The following is a composite of conversations with those two:

"Boy, do I need a second floppy drive! When you boot up Multiplan, it will ask you for the other disk, then ask for the first again, then the other — you have to do 7 or 8 disk swaps before you are up and running. You only have 40K of data available after the program is loaded, about the same as in a 64K Apple II. I already knew it was written in Pascal or C because I recognized that characteristic 'flicker' as it (Multiplan) wrote to the screen. You get the same thing with Pascal using the Apple II or the SAYBROOK board! But that graphics program is not half shabby, and the mouse really works out pretty well. Sure wish the CRT was a little bigger. BOY, DO I NEED A SECOND FLOPPY DISK DRIVE . . . "

## GETCHA PROGRAM HERE!

It is important to keep track of the various Macintosh BASICs. We understand the only one you can buy is Microsoft's busted-BASIC, written in C. Apple has a BASIC, which they are not choosing to make available, it seems. It is almost certainly an offshoot of that very early LISA busted-BASIC, the one carefully busted in Pascal. We just heard that Apple refers to this as a "learner's BASIC." "Learner's BASICs" don't have to run fast, right? The third entrant is Microsoft's announced but unavailable UNbusted-BASIC. Lots of folks are confusing this one with the busted-BASIC, the excruciatingly slow BASIC that you can actually buy today.

(Proving that nothing is as simple as it seems, we have heard from two sources that the very fast 68000 BASIC, the one which compiled each line as it was entered and which InfoWorld attributed to Microsoft, is actually being developed inside Apple Computer. On the other hand it does appear that Microsoft IS working on an unbusted 68000 BASIC to replace the busted-BASIC it now sells for the Mac. Here is how we put this together: both Apple and Microsoft are busily developing an unbusted 68000 BASIC for Macintosh. Both of them are 'compiling' each line as it is entered.)

(We know another company which is developing a 68000 BASIC like that [guess who?]. We wonder who will be done first?)

## AS THE CHIPS TURN

This thing is almost like a soap opera. Having kicked Microsoft out of bed, IBM is showing signs of climbing into bed with Digital Research. Is it a coincidence that Microsoft deliberately busted its non-IBM software in C (much earlier, in Pascal) while it was in bed with IBM? Is it a coincidence that they announced a forthcoming UN-busted 68000 BASIC about two microseconds after IBM kicked them out of bed (by supporting a Unix clone that was not XENIX)? Is it a coincidence that Digital Research is announcing adherence to other languages busted in C just as it is climbing into bed with IBM? Maybe DRI isn't goofing by accident at all!



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The funny thing is, DRI is simultaneously trying to climb into bed with IBM and AT&T! Talk about strange bedfellows! (No, we are not making this up, and we are not describing the plot of an episode of "Dynasty.")

## MASS-MARKET UNIX

We just got a DTACK newsletter renewal from a guy who works for one of those outfits that make very good, and very expensive, Unix boxes. Instead of printing his letter, we are printing our (lightly edited) reply. For one thing, he admits that his company's Unix manual is not 1,000 pages long but 3,000! And he admits they do not like to publicize that fact. Also, we are getting tired of that "we" used by royalty and by editors.

"Welcome back! I thought you were mad at me. When you get issue no. 29 you may recognize yourself on page 23, column 2. The system illustrated on the front cover uses a math processor that is almost as fast as [company deleted], and a whole bunch cheaper!

"FNE originally meant Faithful Newsletter Editor. If you don't believe it, take a look at the number and size of the newsletters I have turned out! I composed the subscription renewal letter on an Eagle II and turned it over to my secretary to retype on letterhead. Then I didn't notice that 'Eloi' was misspelled (you are the second person to call that to my attention).

"You should like Unix—Unix is an absolutely perfect tool for a highly motivated, full-time computer professional who has the cash (or whose boss has the cash) to buy a really good Unix machine. For $37,000 I certainly hope that your system has good response! I happen to get (I believe) equally fast response from the Eagle II I am keying in on at the moment — and you can buy an Eagle II for under $2,000. Where is a Unix box that will give good response and cost under $2,000?

"At $37,000 the Unix box on your desk (which we both agree is very good) is in no way targeted at the mass computer marketplace. When I write that Unix ain't gonna make it, I am referring to the mass computer marketplace. Have you noticed all the IBM PC owners discarding PC-DOS and rushing to by PC/IX? Neither have I. And although you and I appear to have similar sentiments regarding one-man-one-CPU, I just read, after breakfast, a prediction by one Scanlon (I'll send you a copy) which claims just the opposite trend will soon occur.

"A mass market personal computer is going to be a cheap computer, or at least a relatively inexpensive computer. It is not clear to me that virtual memory can be properly implemented on a mass market computer.

"I have a neat little file system I have been building since 1961. Right now I am in the process of filling a fourth four drawer cabinet. Last year when our project engineer was still in school, he needed some information on Hall effect devices. So I reached into the top drawer of file #1 and pulled out the Hall effect folder and handed it to him. Nearly all of the articles were from 1964, because that's when interest in Hall effect devices peaked.

"This relates to your assertion that people in the computer world are notoriously antagonistic to taking measures of performance. Voila! Enclosed is a copy of a 1972 DATAMATION article on 'The Profit in Bad Software.' The GAO is trying to get the Air Force to sell off its CRAY-1 in New Mexico. It seems the beast is only being utilized 55 percent. We now segue to Minnesota, where CRAY is busy reporting that it is going to adopt Unix! I wonder why . . . "

— Felgercarb E., Santa Ana, CA

## POINTS OF VIEW

Your point of view heavily influences your opinion. We all know about the blind men and the elephant. This newsletter is written from the point of view of folks who own, or want to


**IBM SIZE BINDERS, SLIP CASES, PAPER**

- D-ring binders to exactly match IBM's cloth. Several colors. Only $5 each.
- Matching slip cases. $5.50 each.
- Portfolios for games, utilities, etc. Clear plastic pocket on front for program and company names. Pockets for disk and manual inside. $3 each.
- Floppy disk pages for binders. Three pockets. Special static free, no-stick material. Better than IBM's. Only $1 each.
- Quantity discounts available.
- Custom imprinting available.
- Continuous paper fits IBM Manuals 5½Wx8½H. 3 large holes turn easily over D-rings. Sturdy 20# paper. Only $40 for 2500 sheets. 5 or more, $35.
- Floppy disks - excellent quality BASF DSDD, 5¼ disks in plastic easel boxes, with TYVEK envelopes, labels. A SUPER DEAL! $25/10
- Tyvek envelopes, plain white. 100-$15, 500-$50, 1000-$85, 10,000-$700. Printing available.
- Disk duplication service available. 50 disk minimum. Prompt service. Copy protection.
- Function key cards fit around the keys. You write the purpose of the key on them. Only $5 for 20. Custom printing available.
- Micro-perfed 20# paper, tears to 8½x11. 2500 sheets only $27.50.
- Call or write for CATALOG.
- Plus shipping. Minimum order $50. Add 6.25% sales tax in Illinois.

**ANTHROPOMORPHIC SYSTEMS, LIMITED**
Dept. PJ • 376 E. St. Charles • Lombard, Illinois 60148
(312) 629-5160




PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

own, a high-performance computer for their own use. Folks who will likely pay for that computer out of their own pocket. Most of these folks are employed in some field other than the computer industry and hence are not full-time computer professionals (DSEx founder Jeff H. is a pediatrician, for instance). Only intelligent, highly motivated full-time computer professionals are ever going to be able to master Unix, whether the manual is 1,000 pages long or 3,000 pages long.

Let us now examine Unix from the point of view of the guy the letter above was addressed to: he has a $37,000 Unix box sitting on his desk and he does not share CPU cycles with anyone. He is, obviously, a full-time computer professional. The $37,000 came out of his employer's pocket, not his own. He is obviously intelligent because corporations do not place single-user $37,000 Unix boxes on the desks of dummies. You can see how this guy would like Unix.

There is a newsletter about the world of full-time computer professionals working with very good, very expensive (from our point of view) Unix boxes. The newsletter is titled "Unique" and is edited (written) by David Fiedler, who also serves as editor of Unix REVIEW. From the point of view of "Unique," Unix is a very good operating system indeed! And $37,000 is really a moderate price for your employer to pay for a very good Unix box.

Look, Gucci sells $3,000 purses on Rodeo Drive in Beverly Hills. Gucci does not have any shops in the Bronx or in East St. Louis or in Arab, Alabama.

Let us leave Rodeo Drive and return to Santa Ana, where folks are not full-time computer professionals and they buy their own computers. Do you realize that there is a virtually unanimous opinion on Rodeo Drive that they are going to come down to Santa Ana and sell us some $3,000 Gucci purses? That about 100 companies have invested big, big bucks in (mostly 680X0-based) Unix systems which are all going to be highly successful in the mass marketplace? That Unix is going to become the de facto standard 68000 operating system?

How can they possibly believe that? Because of their point of view, that the full-time computer professional looks at the $37,000 Unix box on his desk and the 3,000 page manual sitting beside it and thinks, "Yeah, this is great! Everybody ought to have a system like this!"

The other day we called Jeff H., the DSEx founder, at about 7:15 P.M. Texas time. His wife gave us his work number. We figured he was taking care of some paperwork. Uh-uh. We couldn't talk to him because he was with a patient! He returned the call after 7:30 P.M. and was going to see another patient after our brief chat. The guys on Rodeo Drive look at Jeff's pediatrician income and his interest in computers and say, "We're gonna sell that guy a Unix box!" The truth is, Jeff can afford a Unix box and he is probably interested, having read a lot about Unix.

But as soon as the Jeff H.'s of this world learn about that 1,000 page manual and how long it takes to master Unix, they lose interest REAL fast.

Here is the point which the folks on Rodeo Drive can't seem to figure out: the reason Unix will NEVER succeed in the mass marketplace has nothing to do with the price tag. It has EVERYTHING to do with the time and dedication required to master the system. In the mass computer marketplace folks are pediatricians or engineers or economists or dentists or math profs.

The reason the folks whooping it up on the permanently stalled Unix bandwagon cannot see this is that they have a different point of view (they have hold of a different part of the elephant). Every single person on the bandwagon is a dedicated full-time computer professional who thinks Unix is WONDERFUL — as it really IS, for them!

## DOUBLING FOREVER?

Some folks are beginning to assert that the yearly doubling of the personal computer market is going to taper off soon. For the record, we think that the three-piece computer market will taper off before the personal computer market — there are a lot fewer businessmen than there are situations into which a personal computer can fit. Further, there will be considerable upgrading in the home market as folks learn enough about their electronic doorstops to know that they can do a lot more with a real computer.

We think the doubling-per-year will continue for two years, at least, after the personal business computer market flattens. There are some signals, faint ones as yet, emanating from the IBM PC marketplace which suggest that that market is approaching saturation. It is going to be VERY interesting this fall to see if Big Blue is on target to triple its sales in 1984 over 1983 — and it was an IBM honcho, you will recall, who made that (very uncharacteristic) assertion before a bunch of financial analysts. Remember: THIS FALL!

## HIGH-END UNIX BOXES

Having disposed of personal computer Unix, it is time to look at Unix in its natural habitat — the $20,000 and up systems,



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

either minicomputers or supermicros. You will recall the predictions in 1982 that vast numbers of Unix boxes were due to be sold in 1983. We called those predictions ridiculous at the time, leaving unspoken the thought that they were also irresponsible. Well, it is now 1984 and we can look back and compare those predictions with the facts.

But we don't have to: three other folks have suddenly done it for us! Brian Boyle, in the new publication UNIX/WORLD (not affiliated with either InfoWorld or ComputerWorld), points out that one forecaster published, in a respectable journal, a prediction that 600,000 (six hundred thousand) Unix boxes would be sold in 1983! (We missed that one but caught the prediction of $3 billion in XENIX licenses.) In fact, Boyle suggests, about 25,000 Unix boxes were sold in 1983. The Unix market is a good, solid "B" market with "A+" expectations, he continues. But EVERYBODY — the companies, investors, distributors — all had geared their plans for "A+" results. When what they got was "B", company ledgers bled red ink all over, executives got fired (he specifically mentions Gary Friedman of Fortune), and stockholders filed suit over misleading forecasts (Wicat, Fortune and others).

Geoff Lewis in, of all places, Mini-Micro Systems magazine, March 1984, p. 101, has a six-page article entitled, "Whatever happened to the Unix supermicro boom?" According to Geoff's figures, there were 23,000 micros and 5,500 minicomputers running — not sold, but running — in 1983. This suggests that even Brian Boyle's figures might be slightly optimistic.

In Electronic Engineering Times, March 12, 1984, p. 80, there is an article on supermicros by Omri Serlin. Serlin devotes 40 column inches to debunking Unix, saying pretty much the same things we have been saying here all along. This makes three articles that have suddenly appeared in the mainstream (nonpersonal) computer journals, asserting that Unix ain't wearing any clothes.

## FNE = VERY POLITE

It's a little late, but if Unix actually sold 18,000 to 25,000 [high-end] boxes in 1983, how many are going to be sold in 1984? Twice that number seems a reasonable, or perhaps even mildly optimistic, guess. Now, count all the folks making Unix boxes (count FAST, more are appearing daily). Divide the number of boxes that are going to be sold by the number of vendors and tell us how many of those vendors are going to be profitable. [This is no fun at all. What's fun, as Brian Boyle suggests, is for a wild-eyed foaming-at-the-mouth Unix booster to jump up on the soapbox and boldly predict that "one million — no, make that TWO MILLION UNIX BOXES ARE GOING TO BE SOLD IN 1984!]

## PLAYING THE IBM CARD

Since the depressing facts about the actual sales of Unix are becoming apparent and since it does not seem possible to sweep those facts under the rug, the wild-eyed Unix boosters are looking to IBM as a savior. After all, IBM is omnipotent [right?] and has laid plans to sit everybody in those corporations at one of the terminals of a new multi-user system (Popcorn, iAPX 286-based) running IBM's variant of Unix, plugged into an IBM mainframe.

Sigh. We have news: IBM is NOT omnipotent, as anybody who has observed that company for more than the last five minutes can attest. It makes fairly frequent small mistakes and even (rare) massive blunders of company-threatening magnitude. If you do not believe us, look up what happened to the line of mainframes which IBM introduced just after the 360 series and just before the 370 series. Hint: by comparison, LISA I was wildly successful!

If IBM really thinks it is going to chain comptrollers and sales managers, etc., to a Unix terminal, that will be one of their frequent small mistakes. For starters, a lot of those folks have already had a taste of freedom with a genuine personal computer with genuine instant response and the freedom to select one's own software. Then there is that huge manual — but isn't this where we came in?

*THE FOLLOWING TRADEMARKS ARE ACKNOWLEDGED:* Apple, II, II+, IIe, soft, LISA: Apple Computer Co. Anybody else need a 166th million ack, have your legal beagles send us the usual threatening note. (To the best of our knowledge, Apple has not trademarked Macintosh yet. We don't have a THING planned for tomorrow; we just might mail something to the trademark registry office just for the hell of it. With trademarks, it's first come, first served.)

SUBSCRIPTIONS: Beginning with issue #19, subscriptions are $15 for 10 issues in the U.S. and Canada (U.S. funds), or $25 for 10 issues elsewhere. Make the check payable to DTACK GROUNDED. The address is:

*DTACK GROUNDED*
*1415 E. MCFADDEN, STE. F,*
*SANTA ANA, CA 92705,*
*(714) 835-4884*
□



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## WRITING A CUSTOM ROM BIOS FOR PC-DOS/MS-DOS MACHINES

An interview with Neil Colvin, President of Phoenix Software, authors of a new ROM BIOS for PC compatible computers running MS-DOS.

PJ: What convinced you to write your own ROM BIOS for PC compatible computers, knowing that a BIOS with complete IBM PC function compatibility might incur the wrath of IBM?

Neil: Well, we have been doing system level software for a long time. It is the business we are in. We have done most of the MS-DOS implementations for non-compatible MS-DOS machines in the last three years; for instance, the DEC Rainbow, the Otrona Attache, Seimens machines, etc. This has been a major part of our business for some time. We got to the point where we had, off-the-shelf, all the modules necessary to put together an MS-DOS port for a manufacturer in a matter of two or three weeks, such as modules for a floppy disk controller, or CRT controller, or serial port device, and so on.

Then, besides the requests from non-IBM PC compatible machines, we were being asked for MS-DOS ports from compatible machine manufacturers.

PJ: And this eventually led to your ROM BIOS project?

Neil: Right. Our next conclusion was that DOSs for PC compatible machines are essentially the same. They are all PC compatible, so they must have ROMs. We already had a number of PC-DOS/MS-DOS operating system ports we could pull off-the-shelf for any customer. Consequently, any manufacturer with a PC compatible machine, or a machine with a PC compatible ROM, could buy our DOSs. Several of our OEM customers purchased our DOS products. Then we realized that it would be even nicer if we could sell them the ROM BIOS, too. We sort of worked our way into it from the outside.

PJ: Then the real concept resulted from a need for customer support?

Neil: We had already done PC compatible ROMs for non-PC compatible computers (i.e., non-PC hardware), as part of our manufacturers support business. We did at least three of them, custom ROMs. But when the issue came up about doing a PC compatible ROM for a genuine PC compatible computer, we weren't just in the custom ROM business anymore. We began an "attorneys of the United States" support business. We actually spent three times the amount of money on lawyers that we did on developing the PC compatible ROM BIOS.

PJ: Does your lawsuit insurance cover legal costs as well as possible damages?

Neil: It covers any costs associated with claims concerning copyright infringements involving our ROM BIOS software. The Hartford Insurance Group covers us.

PJ: How did you structure your ROM development process to minimize any copyright infringements?

Neil: The programmer who wrote the ROM BIOS had never seen an 8086 in his life and, consequently, had done no 8086 programming whatsoever.

PJ: Was that an advantage?

Neil: Legally, we felt that it was necessary. We felt safer by having a programmer who had written only TI-9900 code. If anyone ever claimed that he had, at some point, walked into a room with a DOS manual, we could verify that he couldn't read 8086 assembler. He had never even seen the chip before. The first part of his project was learning the 8086. If you look at the code that he wrote the first time around (the ROM has been rewritten probably three or four times now), it looked like he was still programming in 9900 assembly. There is no doubt about this in anyone's mind when they look at the code. As part of our audit trail, we have listings from every week over the life of the ROM code development process. We currently have seven volumes of documentation for prospective customers to look at. It shows how the software was developed and the legality of it. The programmer began with no knowledge of the 8086 and learned it as he went along.

PJ: A professional programmer can write FORTRAN in any language.

Neil: That's right, but how he does it is different.

PJ: Hmmmm. . . . What is copyrightable in regard to a ROM BIOS? Is it the function name and/or parameters involved?

Neil: We actually have an 80-page spec for the ROM BIOS that the programmer worked from. Here is a service routine, here is what it should do. A functional specification for the complete ROM. This is a device service routine for the CRT, which is called by an interrupt. These registers will have these parameters in them, and these error conditions will be returned.

PJ: Are the specs copyrightable?



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Neil: No, they are not. Functional specifications are not copyrightable.

PJ: Only the source and object code are copyrightable.

Neil: If someone were making a plug-compatible machine for a computer, the interface would not be copyrightable. This concern has been decided many times in numerous court cases. "Reverse engineering is legal," in other words.

PJ: If we could all agree upon the specifications for a set of graphics functions then all of our code would be transportable.

Neil: We have done this same kind of ROM software before, but for different hardware entirely, so there has not been a question of copyright infringement. Probably the most dramatically different machine was the Otrona 816. We did both the hardware and the software for it. It is, in fact, a Z80 machine with an 8086 plugged into a parallel port. All the I/O on the machine is done through the Z80, including the graphics. It will run Lotus out of the box. That is probably the most extreme case of a PC compatible environment built around a non-PC compatible machine. It is an architectural design that we are using for a couple of manufacturers right now that I cannot name. They have non-8086 machines but want to add PC compatibility to them.

PJ: Once you have this 80-page specification book and give it to two different programmers, won't the final code, written tightly, be so similar that you might worry about copyright infringement anyway?

Neil: Very little. The modularization will be dramatically different. What they put in each subroutine will vary. Their own internal calling sequences will be different. The way they actually do the functions. There are many approaches to doing video functions. Scrolling the screen can be done in a number of ways.

If you are initializing a CRT controller chip, there are only certain bit patterns you can output to a port. It has already been stated in court, even in the Franklin versus Apple decision, that if there is only one way of doing something in a machine language program in order to perform the function on a given piece of hardware, that particular section of code is not copyrightable. If you output four bytes in a row to a chip to do serial I/O, the fact that you have the same code is not an infringement.

PJ: Could the programmer who wrote the ROM BIOS code be independently sued?

Neil: No, unless the signed affidavits that he submitted stating that he had not read an IBM DOS manual are invalid. Aside from that he has no liability.

Our copyright infringement insurance policy covers everybody, including our customers who purchase the BIOS from us.

We feel there will never be any legal involvement with IBM regarding the BIOS. In all their years of business, they have never initiated a suit that they have lost. OK. We have such an overwhelming amount of evidence on our side to prove that our BIOS is not infringing. There is little chance of anyone challenging its originality. Part of the insurance is to cover any legal intimidation that may occur as a result of the product.

PJ: Who designed your non-infringement strategy for the BIOS development?

Neil: We consulted with a number of legal councils including some who were involved with the design of Compaq's ROM. There were about five different legal firms involved who conferred and offered opinions. We drew up procedures and documentation standards that seemed to meet everyone's standards. We had an overwhelming weight of legal opinion on our side.

PJ: What do you call your BIOS product?

Neil: We call the entire product our "Phoenix PC Compatibility Package." There is a ROM BIOS part, a BASIC part, and so on.

PJ: What do you think the potential market is for the package?

Neil: When we started, we were only thinking about computer manufacturers and figured around 15 buyers. But already we have had one self-service gas station pump manufacturer come to us. He is changing his station computers to be PC compatible so the machines can do accounting and also read the figures from the gas meters. We are now expecting around 50 to 60 customers, including process control manufacturers. We talked to one person whose computer runs milking machines in the mid-West. They decided to



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

make the next version of their machines PC compatible. We are seeing vertical market companies either OEMing their equipment or designing it from scratch to run PC software. Their customers want to gather their business data and run packaged software, all on the same machine. These customers never advertise in BYTE or normal computer magazines. They would only be in dairy journals and the like.

Honestly, we have one customer whose representative we have met with three times but still don't know who we are dealing with. He has not informed us which company he represents. It is a very strange marketplace. What I'm saying is, people have come to us whom we would never have found by using normal marketing channels. The number of companies making PC compatible machines for very different reasons is much larger than anyone in the industry suspected.

PJ: The fact that you can supply MS-DOS, BASIC, and the ROM BIOS in one package is tremendous. Previously, a company would have had to write at least the BIOS from scratch to get the job done if they wanted a MS-DOS machine. Now we can all run the same functionally compatible software.

Neil: We are aware of at least one company that decided to enter the PC market only because of the availability of our software; a large electronics firm.

PJ: Carrying things a step further, is your BIOS bringing a more important compatibility and stability to the market than even MS-DOS? Has your 80-page spec manual and availability of the BIOS finally brought everyone up to an equal level?

Neil: MS-DOS is a standard by itself. The ROM BIOS could be considered a standard, too.

PJ: Are packaged software products your main business?

Neil: About half of our business is consulting. We have been doing it for over five years now. We work with manufacturers at the design level in both hardware and software. Take Otrona 816, for instance; we did the whole design, including the prototype PC boards.

Our software packaging grew out of the ongoing need to supply products to various customers. In one case, a manufacturer already had a machine, then came to us. We flew a person down to them, plugged our ROMs in, gave them a disk, and they were up and running a PC compatible environment in three hours. That's what you call "off-the-shelf."

PJ: How many employees do you have?

Neil: We have 20 regular employees and about 20 consultants who are on call. About 9 people are programmers, 3 people are hardware, and the rest are administrative. Our return per employee is one of the highest in the industry. The ROM BIOS package was one of the first that we actually advertised. We have never had to go looking for business. Most things in the computer support industry are word of mouth, and we are famous for getting work done on time. We don't accept service contracts that we cannot deliver on schedule. Our credibility in the industry has always been rather high. We want to stay in the consulting business. We do have information listing our complete line of software products, should anyone desire a more detailed description of what services we provide.

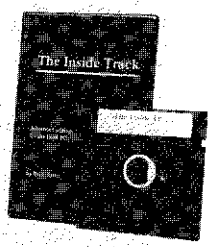
*Phoenix Software*
*1420 Providence Highway, Suite 260*
*Norwood, MA 02062*
*(617) 769-7020* □


# Know Thy PC!

Are you writing programs in BASIC or Pascal? The popular **Peeks 'n Pokes** has a disk with 58 programs and a 38-page manual that helps you get 'underneath the covers' of the PC. Learn how to use PEEK, POKE, INP, OUT, and DOS/BIOS function calls to do what you want, fast! Do you want to perform functions not available from BASIC or Pascal? It's all explained in the manual and demonstrated in the sample programs. Source code included!

**Peeks 'n Pokes** shows you how to:
- Access the system's configuration
- Unprotect BASIC programs
- Scroll part or all of the screen
- Access the file directory
- Logically swap printers
- Read and change the keyboard
- Find more Peeks and Pokes
- And much more... for only $30.00

Want to know more? **The Inside Track!** is a collection of advanced utilities for the PC programmer. It contains a disk with 61 programs, a 42-page manual, and a fold-out memory map that helps you get better performance from the PC. With this package you can give your programs assembler-assisted speed from high-level languages, get control over memory, customize and control the PC, and more. Some programs require DOS 2.00. Source code included!



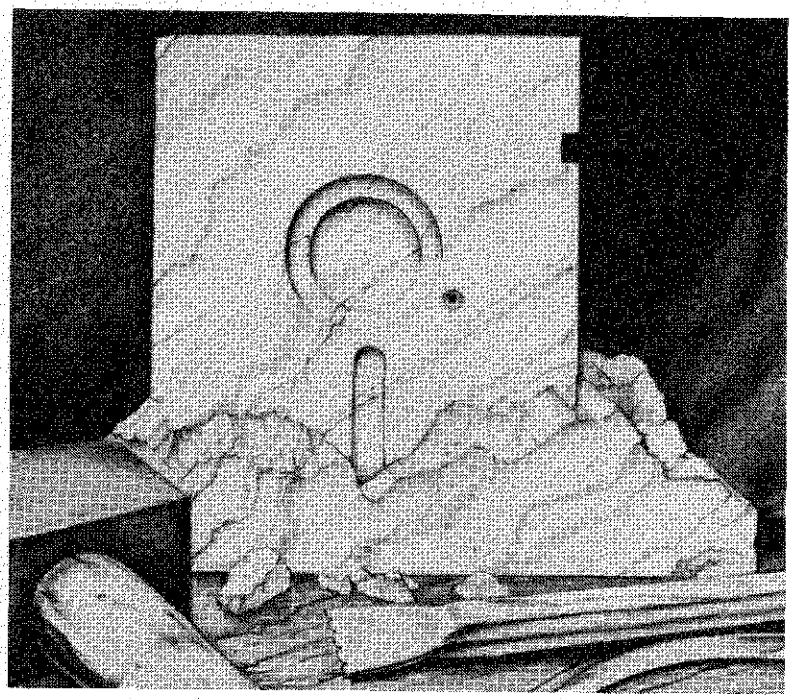

**The Inside Track!** shows you how to:
- Read/write files as fast as DOS
- Display data on the screen faster
- Reserve memory for your use
- Copy memory to another location
- Copy-protect your programs
- Load large programs faster
- Control the keyboard settings
- And much more... for only $45.00

MasterCard and VISA accepted. Shipping charges: $2.50 per order for UPS; $2.50 per item for First Class Mail to USA and Canada; $6.00 per item for Air Mail outside USA and Canada. Dealer inquiries invited.

Data Base Decisions • 14 Bonnie Lane • Atlanta, GA 30328 • 404/256-3860




PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



# *"In the art of programming the difference between greatness and mediocrity is often the quality of the artist's tools."*

In every top programmer's tool kit there are usually a few secret weapons. PolyMake and PolyLibrarian have become the newest essential secret weapons for hundreds of professionals who are setting new standards of productivity and performance for themselves and the industry.

## *Praise From Professionals*

"PolyLibrarian is a powerful tool for serious hobbyists and professional programmers. It is a thing of beauty; a work of art. The documentation is excellent, the large selection of commands are intuitive to learn and easy to use, and the program itself provides a service of inestimable value. It is refreshing to see a tool of this caliber available for serious programmers. The design of this program shows foresight and ingenuity. It sets the human engineering standard for programming utilities. If all of a programmer's tools were so simple to work with, better programs could be developed in less time and with fewer headaches. *Bravo, Polytron!* I look forward to your next product."

*Dan Rollins,*
*Programmer and Author writing in PC Age*

"PolyLibrarian is an extensive, friendly utility . . . an excellent tool for serious programmers. It combines professional quality with a flexible user interface."

*Greg Estes*
*Editor, Programmers Journal*

"I am thoroughly pleased.[PolyLibrarian] is definitely one of the best products of its type that I have ever used on any system."
An Unsolicited Comment from
*Steve Kauffman*
*Consulting Engineer*

To Order or Request Literature Call

**1-800-547-4000**

Ask For Dept. 31
VISA & MasterCard Accepted
Or Send Checks, P.O.'s To:
**Polytron Corporation** DS-310-F1
P.O. Box 787, Hillsboro, OR 97123
Add $1.25 Shipping Charge For Each Product Ordered.

## POLYLIBRARIAN

***The Object Module Library Manager for MS-DOS™***

PolyLibrarian lets you create a single library from related Objects Modules. The IBM PC Linker will then select only the modules necessary to produce an executable file. PolyLibrarian can: ☐ Create, organize, reconstruct, dissect, and compress libraries of object code modules. ☐ Add, delete and replace an object module. ☐ Extract a module from a library to create an OBJ file. ☐ Change public and external names. ☐ Optimize a module to shorten overall size of a library. ☐ Modules are time-stamped within the library. ☐ Supports full path names. ☐ Supports Microsoft's original and latest Linker (Version 2.4). ☐ Includes installation program to customize options to your specific needs. ☐ Features three Modes of Operation (Interactive, Command File, Command Line) that can be freely intermixed. ☐ Surpasses performance of other primitive librarians. ☐ Features a highly functional user interface that uses simple meaningful key words. ☐ Requires 128K RAM, MS-DOS (PC-DOS) 1.1, or 2.0. ☐ Compatible with any compiler or assembler that uses the MS-DOS (PC-DOS) Linker. ☐ Shipped with comprehensive Users Manual including step-by-step tutorial for ..... **$99**

## POLYLIBRARIAN II

***Extends PolyLibrarian's power to Intel™ Format Libraries.***

PolyLibrarian II is an enhanced version of PolyLibrarian. In addition to all the features and functions listed above for PolyLibrarian, this version adds a new dimension of productivity for programmers using Intel Language Products. ☐ Compatible with any Intel Format Library and any Microsoft Format Library. ☐ Compatible with Intel's Link 86 under MS-DOS. ..... **$149**

## POLYMAKE

***The Intelligent Program Builder & Maintenance Tool for MS-DOS.***

PolyMake frees you from the need to remember which files depend on others and which files have been modified. PolyMake will: ☐ Remember the exact sequence of operations necessary to make a new version of a program. This will significantly reduce development time, prevent bugs & manage large software projects ☐ Automatically invoke your compiler, assembler, linker, librarian, or do whatever is necessary to bring all dependent files up to date. ☐ Compare the date and time of all relevant files and use internal rules to rebuild a program or complete software system. ☐ Rules can be modified and expanded. ☐ Extensive debug capabilities help construct efficient make files. ☐ Includes features superior to UNIX Make. ☐ Fully automatic operation under MS-DOS 2.X. ☐ Full path name capabilities under MS-DOS 2.X. ☐ Requires 128K. Shipped with a comprehensive Users Manual **$99**

TM

## POLYTRON

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

# REDIRECTIONS

## RISC, TRANSPUTERS & MASS STORAGE

*by Frank D. Greco, AGS Computers, Inc.*

Recent software price listings sent by Microsoft to several of their U.S. OEM dealers include a couple of interesting items: MS-DOS 2.5 for $60, and MS-DOS 3.0 for $100. No details are included in the price list, unfortunately. Another interesting tidbit: the price list sent to Microsoft's Japanese OEMs includes two other software offerings: MS-NET and MS-UX. One would logically guess that MS-NET is their PC networking software, but MS-UX is described as a high-powered operating system based on Unix. Where does this OS fit in the Microsoft scheme of things? And why list this product only for the Japanese?

Speaking of the Japanese, expect to see PC/IX running on the Fujitsu 8086-based micros. It might not be called PC/IX (IBM owns that moniker), but it's the Interactive System's port of System III Unix. Apparently, a MAJOR company is interested in porting Unix to the Fujitsu Micro 16s series. According to information from Fujitsu's Professional Microsystems Division (PMD), the Micro 16s micros are currently based on the 8086 as the main CPU, Z80A as a CP/M driver, and a 6809 as the video controller (640 by 200 in 8 colors). However, what is interesting about these machines is that if you want more horsepower, unplug the 8086 board (it's not on the motherboard) and plug in Fujitsu's 68000 board. Add Xenix and a hard disk, and bingo . . . another Unix box.

So far there are five Unix shows slated for 1984. The January Uniforum conference in Washington, D.C., is past history. The Usenix user group will have their conference at Salt Lake City during June. This group has been primarily a technically oriented group; however, this year IBM and AT&T will have massive booths at the conference. The "First Market Forum for Unix" is going to be held at the Hyatt Embarcadero, San Francisco in May. This one is sponsored by Yates Ventures and BOSCOM. Call 800-343-0974 for details. Unix Systems' Expo/84 is going to be held in Los Angeles, September 11-14. Call 617-965-8350 for details. The last one is called Unix Expo and is going to be held October 16-18 in the Big Apple (no . . . not inside of a Lisa!). The conference will be held at the Sheraton Centre Hotel, NYC, and the exposition will be at the Marina Expo Complex, NYC.

I was also informed that the Comdex/Winter show, this year in Los Angeles, April 5-7, had a plethora of Unix vendors, as well as the never-ending entourage of PC-compatible manufacturers. Another quasi-show of note to Unix aficionados in New York is the April 25th Unigroup meeting (East Coast Unix users group). This gathering will showcase "Unix on the IBM PC." Various vendors (including Big Blue) will each give a 30-minute presentation on their port. Benchmarks will be run, using each vendor's product, as the grand finale of the meeting. I'll publish the benchmarks in the next REDIRECTIONS column.

On March 27 at 10:50 a.m., AT&T formally announced their entrance into the computer business with their 3B product line and System 5.2 Unix. Of interest to the microcomputer buff is the 3B2, AT&T's low-end machine. It uses the 32-bit WE 32100 CMOS microprocessor and costs approximately $10,000. For more information, call AT&T's 3B hotline, 800-833-9333.

RISC-BASED COMPUTERS

While we are on the subject of AT&T's press conference, Jack Scanlon (AT&T Technology's V.P. of Computer Systems) mentioned in passing that "computer systems and computer technology are ever expanding, like the new RISC-based computers." Why did Mr. Scanlon single out that particular technology? Why didn't he mention things like improved user interfaces, windowing, 32-bit microprocessors, gallium arsenide/emitter coupled logic technology, optical disks . . . the list goes on.

Well, my curiosity was spurred by that comment, so I did some research on RISC-based computers. First of all, for those of you who do not know what RISC means, it stands for Reduced Instruction Set Computer. Long an academic project in schools like Berkeley and Carnegie-Mellon, RISC-based computers have fewer instructions (machine instructions, that is) and more registers. The physical space that is usually reserved on-chip for logic circuitry (i.e., the implementation of machine instructions like ADD, INC, and JUMP) is reduced to create more registers. More registers available to a given program, like a compiler or an interpreter, usually means faster execution. Pyramid Technologies has a Unix-based RISC computer that operates three to four times faster than a DEC VAX 11/780 (for CPU intensive programs) but costs about $100,000. It doesn't need special flooring or air conditioning, either.

If you have been reading the trade papers lately, AT&T has been making a bid to buy INMOS, the British semiconductor manufacturer. INMOS makes a little device called a "transputer." Samples will be available during the last quarter of 1984. A transputer is a 32-bit RISC-based microprocessor that is designed for multiprocessor systems. Communities of processors, each tackling a separate task, help alleviate the Von Neumann uni-processor bottleneck. Each (!) transputer in a multiprocessor system operates at 10 MIPS (million instructions per second). This device is called a transputer after the words "transistor" and "computer," since it is both a computer on a chip and a silicon component like a transistor. I would guess that AT&T, besides needing a modern manu-



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

facturing plant to make the chips, could use the transputer as future leverage against any "ordinary" 32-bit offering from IBM (i.e., iAPX 386); it also would be substantial ammunition against the Japanese with their PROLOG-based multiprocessor systems. These transputers can be used in truly concurrent multiprocessor systems. Certain systems lend themselves quite naturally towards a network of transputers: applications like artificial intelligence, high powered graphics systems (solid modeling, real-time animation), fault tolerant systems, database systems, and — guess what else — telephone switching networks!

INMOS also has a programming language that takes advantage of their transputers; it's called Occam. Like Modula II and Ada, Occam has concurrency built into the language to make the multiprocessor system design "straightforward." If you want to evaluate Occam, INMOS has an Occam Evaluation Kit that runs on the IBM PC, VAX, Apple II, and Victor 9000.

Everyone knows that AT&T bought a substantial portion of Olivetti to gain a footing in the European marketplace, but everyone does not know that Olivetti is a major investor in the Mindset computer. Mindset is the graphics computer that has been widely publicized of late.

NEC is keeping busy these days by signing on as a second source for Zilog's Z80000 (Z80k) 32-bit microprocessor AND developing a new memory device capable of storing one gigabit. The memory device will initially be available in 16 and 64 megabit versions in two to three years. These chips are similar to bubble memories but differ in that they use the twists in the magnetic domain walls that surround bubbles. NEC is calling these twists "Bloch lines." Since the access time of this device is comparable to bubble memory, which is relatively slow (50 milliseconds), it will be relegated to external storage. NEC has said that the physical form of the device when it is marketed will measure 0.4 X 8 X 1 inches.

PRINTERS

There have been a bunch of dual mode (letter quality and dp quality) matrix printers hitting the market of late. I'll describe some interesting ones. The Epson LQ-1500 printer, which prints 200 cps in draft mode and 67 cps in letter mode (Epson doesn't call it "letter quality" mode for some reason), has a 24-pin printhead (the ordinary matrix printer has 9 pins), and 3 different character sets in ROM with 128 characters that can be downloaded. Only proportional character sets can be printed in letter mode for some reason. The Epson also has high-density bit graphics mode for up to 240 dots per inch. Of course, the escape sequences are compatible with the older Epsons. This is a very nice printer, but you can still tell that the print came from a matrix machine.

Another dual mode printer is the Toshiba 1350 24-pin matrix printer, which lists for approximately $1,900 ($1,650 retail). It is more impressive than the Epson LQ printer. This is the printer that Tandy is OEMing for use with their micros. I saw a sample of the print from it and was thoroughly impressed. I had to strain my eyes to see any difference between it and a quality Daisywheel print. Some fonts were so good that they fooled me, and I am VERY critical. The Toshiba 1350 runs about 192 cps in draft mode and about 100 cps in letter quality mode. It comes with a standard Centronics interface, but you can get a RS-232 serial interface if you need it. Toshiba even has a lower cost printer that has most of the features of the 1350: the Toshiba 1340. They are marketing the 1340 for low-end machines like the PCjr and the Apple.

Fujitsu has announced a nifty matrix printer called a DPL24 that also has a 24 pin print head. At $1950, the cost is similar to the Toshiba 1350. Draft mode really flies at a speed of 240 characters per second. Letter quality mode is about 80 cps. The DPL24 comes with several fonts: Courier 10, Prestige Elite, and some boldface fonts. Fujitsu is really coming on strong in the American market. In addition to their printer products, they announced an increase in their ownership of Amdahl Corporation to a significant 49.5 percent. And you thought that Amdahl was an all-American company!? Fujitsu also announced an electroluminescent flat panel that displays 24 lines by 80 alphanumeric characters, AND 320 by 256 pixels in graphics mode. Their labs also announced that they are working on an ink jet printer that offers 1000 dots per inch; that's almost photo quality! Fujitsu is one of the few companies in the world that IBM does NOT take too lightly.

But back to printers . . . The most unbelievable dual mode printer that I've seen so far has been the Advanced Matrix Technology, Inc. (AMT) matrix printer. Draft mode is a respectable 250 cps (nice only for debugging purposes), but the letter quality mode (they call it "better-than-letter-quality" mode) prints at about 40 cps. The print in letter quality mode is unbelievable! This printer was created by some of the original designers of the Malibu printers. (Remember them? They started this double-purpose matrix printer hoopla.) Graphics mode goes up to 240 vertical by 720 horizontal (.0014 inch addressability). It comes with several high-quality fonts and many optional fonts. An interesting option is a four color ribbon that is software controlable. Through artifacting, you can get a whole series of colors. Believe me, the letter quality HAS to be SEEN to be believed. The price is about $3000, unfortunately. The only question is the stability of the company. Will they be around tomorrow for support? Malibu folded quickly; let's hope that this company sticks around. This looks like a super printer.

UNIX/UNIPRESS/APPLE

With Unix starting to take off in the microcomputer industry, Apple is supposedly designing a new version of the Macintosh (or Lisa, whichever way you look at it) with an ear bent towards this flexible operating system. According to rumors,



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

it will have a memory management unit for Unix type operating systems (it should have had an MMU to begin with!), accept more RAM, and have a color monitor option. These rumors have Unisoft (68000/Unix experts), Unipress (a distributor/developer of Unix for Lisa), and Apple putting their collective heads together to formulate a Unix strategy. Either Mac will be upgraded or the Lisa will be redesigned. I would bet the latter, especially with some new blood on the Lisa staff nowadays AND the market niche for a high-end machine that lies on the Lisa side. Jobs has been quoted as saying that " . . . Unix is a pretty lousy operating system to put inside a work station. It's old technology . . . But we think it's really great in a multiuser environment, and we'll come out with a server . . . Macintosh and Lisa can be used as terminals." One Apple announcement that somehow got lost in all the Macintosh splash was the Apple Laser printer. For $5000, you can have a super laser printer capable of copying any of the neat graphics of the Mac and the Lisa. Since Canon is the only company out there with a laser printer under $5000 ($3000 in OEM quantities), they would be the logical choice as the OEM. Imagine hanging a laser printer in a network of Macs and Lisas with a central unit running Unix!

Bill Budge, the super-graphics programmer, has created a product very similar to MACdraw for the Apple II series. Bill Atkinson wrote the in-house version of MACdraw. Budge's new Apple II look-a-like is called Mousedraw, and it is impressive! It's in color, too! Budge gives credit where credit is due and openly says that he was inspired by Atkinson. Not a bad program for the venerable Apple II line. Expect to see some unbelievable Macintosh programs from the young Mr. Budge; he is best friends with one of the MAC development staff, Hertzfeld.

Don't you think it is ironic that IBM calls the PCjr cordless (infrared) keyboard a "FREEboard" in most of their ads, even though the more sensible users will go out and buy a Keytronics $255 keyboard? If you want a cordless keyboard for your PC or PC/XT a la PCjr, contact Cherry Keyboard Headquarters at 312-578-3524. I heard a price of around $275 for the cordless beast.

MASS STORAGE

Kamerman Labs out in Oregon, 503-626-6877, has a 10-Megabyte Hard Disk System for the IBM PC at $895. That includes a half-height hard disk unit, disk controller board and software. Sounds too good to be true. As with all NEW vendors, it usually pays to wait for a complete review of the product before purchasing. Those of you who bought the early Removable Cartridge Winchesters must know the "wait and see" ballad by heart.

Do you want MASS storage for your IBM PC? Well, National Memory Systems in Livermore, California, has a new 8-inch Winchester system that offers 168 megabytes! The PC-8002-8 comes with MS-DOS 2.0 device drivers, disk controller board, integral power supply and full documentation; all for $9000 in OEM quantities.

Dragon Industries in Hopkinton, Massachusetts, is definitely not dragon behind (couldn't resist the pun). They're offering 65-meg and 140-meg hard disk drives for $4995 and $6995 respectively. Both include controller, power supply, cables and chassis. Instead of an 85 msec average access time (PC/XT), Dragon gives you 30 msec! If you really want to go for the gusto, plug in dual hard disk drives for 260 megabytes of formatted storage.

Kodak has finally entered the computer peripheral fray. Floppy disk and disk drive manufacturers like Dysan, Wabash, Seagate, and Tandon are a little concerned about this giant's entrance. It seems they ought to be. Kodak has announced a 3.3 megabyte 5 1/4 inch floppy disk drive. Yes, you heard that right. This amazing floppy gives you 192 tracks per inch, a proprietary servo system for accurate head positioning, and 3-msec track to track speed. What's more, the Kodak model 3.3 also features automatic and transparent reading of 48 and 96 tpi disks.

ANNOUNCEMENTS/RUMORS

Commodore finally announced their Z8000 machine at the Hanover Fair in West Germany. The machine runs the Coherent operating system, which is a variant of Version 7 (pre-System III) Unix. It comes with two high-capacity disk drives and 256K RAM. It supposedly has extremely high resolution graphics and a number of easy-to-use features. No specifics yet, but I really hope this one has some hardware memory management. The price has not been set, but rumors have Commodore setting its sights on Clive Sinclair's QL machine and Amiga's little 68008 micro. Both of these machines are in the $500 to $1000 range. I cannot see Commodore pricing this machine at such a low level, although I would love to see it happen. Unix fans, don't hold your breath. Commodore has a history of announcing machines to test consumer reaction, and then discontinuing the product if the reaction is bad or nonexistent. I'm not sure if a low-end market exists for Unix.

Commodore also announced another PC-compatible product (doesn't this get boring, guys?). Actually, it is a Bytec Hyperion, a transportable computer that has a small but effective graphics screen and 256K RAM. Their other announcements at the German show dealt with the Commodore 16, a 16K RAM machine capable of being upgraded to the Commodore 264. A logical guess says that the model 16 will supplant the old VIC 20. Several C64 compatible printers and C64 mouse/lightpen/touchscreen software (might as well cover ALL input devices while you're at it, Commodore) were also announced.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Borland International, the people who offer the super $49.95 Turbo Pascal for the IBM PC and PCjr have dropped the $100 additional object code distribution license fee. If you used Turbo Pascal to develop a package for the mass market, you were previously required to pay a $100 licensing fee. This is no longer the case. If you already paid the $100 fee, it will be refunded. Just call Borland directly. When I mean directly, do not call the 800 number. The 800 number is handled by an extremely obnoxious and rude person who actually hung up on me when I asked for a technical support person. This happened on several occasions. I informed Borland about this and was relieved to hear that their answering service was going to be dropped due to MANY complaints from disgruntled customers. Just to clear Borland, this is an answering service and not a company employee. I got the best of help when I called Borland International directly. I'll benchmark Turbo vs. IBM Pascal in a later column.

Strong rumors link IBM and Mark Williams (authors of the Coherent operating system). It seems that IBM is taking an in-depth look at Coherent as a possible Unix (PC/IX) alternative. Coherent only takes up about 1 megabyte of disk space as opposed to PC/IX which needs at least 5 megabytes and up to 7 megabytes (for every Unix utility). Note that Coherent is based on Unix Version 7 and PC/IX is based on Unix System III (System III is more recent than Version 7). It is very interesting that IBM is investigating non-System V ports. Will IBM support System V? With virtually no memory capability and no high-level primitives for file/record locking inherent in AT&T's System V Unix, what do YOU think?

Speaking of Big Blue, IBM quietly increased its stake in Intel to 20 percent. Intel reported record revenues during 1984, and IBM subsequently boosted its interest from 18.8 to 20 percent. According to an agreement between the two, IBM can own up to 30 percent of Intel. Also of note, Intel's 32-bit version of the 8086, the 80386 (iAPX 386), will be manufactured using an Intel variant of the original RCA CMOS techniques called CHMOS (Complementary Hybrid Metal Oxide Semiconductor). Intel told me that the 80386 will not be available, even for sampling, until late 1985.

Since Timex dropped Sinclair distribution, I've noticed extremely low prices for the interesting Sinclair 2068 micro. Some electronic shops in New York City are selling the thing for as low as $50. At that price, it is, as they say, a STEAL.

Digital Electronics Corporation has announced a large, flat-panel display that uses vacuum fluorescent technology. The panel measures 5 inches by 3.75 inches and can display 30 lines by 53 columns (??) of alphanumeric characters. It is also pixel addressable, with a resolution of 320 horizontal by 240 vertical. It accepts an RS449 or RS232 interface and is capable of gray levels! If you want quicker display updating, a parallel interface with DMA is also available.

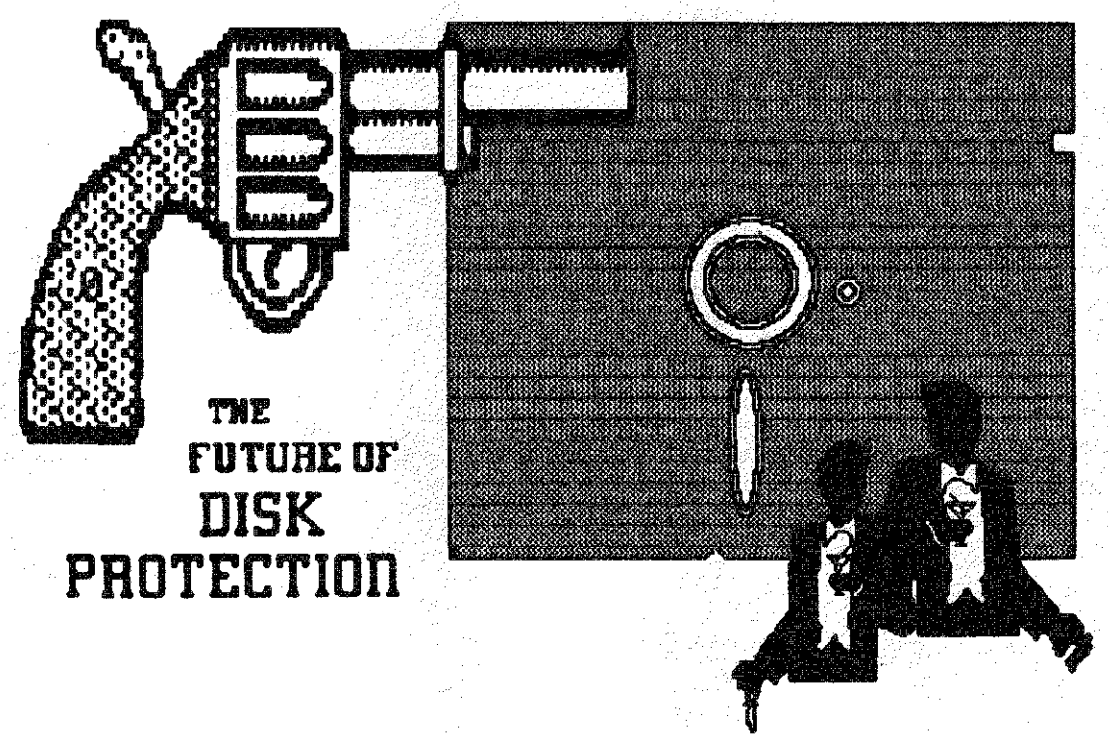
At the Trenton State Computer Fair, I talked to a Sord distributor about the Sord lap computer. If you don't know already, the Sord lap computer is made in Japan (no surprise nowadays for lap computers) and utilizes a Z80 processor. It comes with a built-in 300 baud modem, 2 parallel ports, RS 232 port, empty ROM cartridge socket (for up to 64K ROM), bar code reader port, battery charger, integral microcassette, spreadsheet, data base program, word processor, communications package, and graphics software. Maximum RAM capacity is 64K RAM. It has Liquid Crystal display of 8 lines by 40 characters OR, in graphics mode, 256 by 64 pixels. This vendor was selling the 64K RAM version of the Sord for $800. Pretty amazing little machine, especially at that price (even though the return key is too small). But what we discussed was a new machine that Sord is promising him (and other Sord distributors) in about one month's time: a version of the lap computer, but with a CMOS 68000 and a larger display. Impossible, I said. Motorola hasn't even announced a CMOS 68K to be available in the short term. Well, it seems that in Japan, Hitachi is making the part. A lap computer based on the powerful 68000; now that's something! The next thing you know, they'll put a computer in a watch. You say Seiko already did that? How about a T.V. on your wrist?

For $300 yet . . . . □

– Famous Computer Personalities #158 —
Willard Foote, senior programmer at a word processing software firm spent 42 hours in Cafe' du Mond recently, waiting for a pop-up menu.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED




*by Richard Larratt*

*Editor's note: This article evolved from a phone conversation with Mr. Larratt. We discussed the pros and cons of copy protection — especially the new type of disk-locking schemes.*

*Three software protection methods lead the pack. The oldest method is formatting the diskette with nonstandard sectors. Much of the current software uses this approach even though it is easily circumwented. A second scheme involves writing a completely new BIOS which is loaded alongside application. All system calls use the new BIOS rather than the ROM BIOS. This may work great for a one-program environment, but it plays havoc with hard disk operating systems and software integration (multi-tasking). The newest (soon to be popular?) method is laser burning. A particular section of the diskette surface is burned so any physical read always fails. If the program does succeed in reading data from that area, then the software refuses to function. This laser burning method is about to hit the streets en masse.*

*The software protection issue is a hotly contested one. Are disk-locking tricks designed to protect the software developer, the publisher, or the end user? Are we guarding against copyright infringement or defending big money? As Mr. Larratt points out, there is a definite need for copy protection, but NOT, as you might suppose, after the software finally hits the street . . . .*

I "went public" on the disk copying subject by writing the first advertisement for SAFECOPY (now COPYWRITE), a disk backup program. The ad ran in InfoWorld, vol. 5, no. 25 (June 20, 1983). Later I advertised and sold a programmer's disk utility called DISK EXPLORER, which examines diskettes in any format. That ad is now history and I offer it only as a reference so you'll know where I am coming from.

If you desire more information about any of the topics I'll be discussing, just write or phone. I'll be happy to help you.

COPY PROTECTION

A copy protected diskette is usually a diskette on which the data has been stored in a format different than MS-DOS anticipates. A really simple illustration of that is that DOS 1.1 cannot read the 9th sector on a diskette formatted by DOS 2.0, since 1.1 anticipates 8 sectors per track. In the same vein, DOS 2.0 cannot read a JFORMAT diskette, which in the interest of getting 400 KB on a diskette puts ten 512-byte sectors on each track (40 tracks per side, two sides, hence 80 X 512 = 400 KB).

I can only assume we'll see ten sectors in a future release of DOS. Frankly, this sort of "game playing" really steams me, because if you have a disk-copy utility and look at the tracks, you'll see the "empty" space towards the end of every track (a lot on a DOS 1.1, a bit less on 2.0, and still less on JFORMAT). You can expect that space to get filled and also expect other



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

writers (not this one) to publish breathless articles about how the IBM PC just keeps getting better.

If you are not a "disk mechanic" (I mean that in a generic sense) you may not know that you are running distribution software that puts 18 or more sectors on a track. This confuses the official operating system and causes it to report the event as an error. Copies cannot be made. If it is an especially novel locking scheme, it will also stop programs like COPYII-PC and COPYWRITE.

SIG SOFTWARE

As a result of the prolific use of software protection, I find myself turning more to public domain SIG software. Not only because it is cheaper, but because it is not "cluttered" by bizarre diskette formats or tricky BIOS's.

If the truth be known, most of us "insiders" manage to avoid protected software; i.e., if MicroPro and/or Ashton-Tate put the Prolok (laser-burned) protection on their new versions of WordStar and dBASE II, most of us will stay with our older versions and exchange patches via the SIGs. The older versions can be made full-featured this way. Even PC WORLD and PC MAGAZINE are publishing DEBUG patches of old versions of commercial software which bring them up to the performance level of the current and copy protected versions, advertised in the same issue. Who is kidding whom?

Consider the case of Lotus. A great piece of code, but, as distributed last year, it could not be loaded onto a hard disk. It was only a matter of weeks after its release that a BBS patch allowed you to do it. If you were a neurotic, you bought the copy protected Lotus and ran the other. I understand the current version of Lotus may have laser-burn protection that lets you run off the hard disk, except that you need the "burned" disk in the default (A:) drive.

If only the copy protection would end. If only it could have ended already! This not being the case, COPYII PC and COPYWRITE were ready and able to intercept the BIOS system calls and trick the protection, allowing Lotus "backups" to be made. But not for long. Perhaps by the time you read this, Lotus will be using a unique BIOS (goodbye ROM). So where is the "beef"? The "beef" is that it becomes difficult, if not impossible, to merge "integrated" Lotus with other programs or files in one particular MS-DOS environment. That is a shame, because Lotus is such a fine product.

SHOULD YOU EVER COPY PROTECT YOUR SOFTWARE?

I don't expect to sway you to my point of view, "don't copy protect end-user products," with this one article. Maybe you work for MicroPro or Lotus and are very emotional about the subject of copy protection and software piracy. I still believe copy protection should not exist in the end-user market.

Look at the history. So far, any program that has had big volume sales, such as Lotus or dBASE II (not yet protected) has been broken and/or copied. Maybe a unique operating system can achieve copy protection (disk-format protection has been beaten consistently in the past three years of escalation) but at a high price.

Ideally, you SHOULD copy protect your software BEFORE you go to market. I know, because I've been ripped off as a result of not following this advice. Let me take you, first-time author, through the scenario. You have written a program and you're looking for a distributor (a person or company that says things like "channel programming" and isn't talking about computers). If you get to first base with these guys, they are going to ask to see a runtime version of the software before they will go any further.

Copy protection, at this stage, is a must. I don't mean to malign the reputation of publishers or distributors. I am only saying that the ONE time you don't want your unprotected code out of your control is BEFORE you have signed a distribution agreement.

Do not despair with the knowledge that the best copy utility programs can copy anything on the market. Today they can, but tomorrow they can't. Even with laser burning and trick BIOSs, they are not foolproof by virtue of being automatic. The programmers writing protection schemes for the top twenty products are able to stop the copy utilities with each new release, but the success is short-lived. Copy utility suppliers offer frequent updates to ensure their ongoing viability. For you, first-time software developer, someone who has spent maybe thousands of hours, if not dollars, on your program, test your private protection against the latest release of COPYII PC and COPYWRITE. You can be sure that some "assistant" in a distributor's software submission department will have a copy program for "backup" purposes.

If, gentle coder, you decide to avoid getting ripped off before you go to market, I can recommend two inexpensive products that will do the job: SLK/F & COPYLOCK.

The reason that these two products are feasible is that none of the top twenty products use them for copy protection. Consequently, there is no reason for the utility guys to take the time to figure out their schemes.

SLK/F is $150, COPYLOCK is a bit more. Talk to the authors; they'd probably work with you to help you protect your code. (Actually, the copy utility authors are just as good at writing protection schemes.) Definitely be aware that the value of your protection scheme is directly related to how long you have been using it.

I hope you go to market without even "logical" copy protection. If you lock up your software, the copy utility guys will



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

immediately break it. If you've decided to go the laser-burn, trick BIOS route, I firmly believe it will be the kiss of death except for truly spectacular software. All things being equal, why would anyone buy such a frustrating distribution hassle if there's anything else even close that's in a "regular" DOS format?

Before you go crazy, plan to be a one-product millionaire, buy a laser, or write trick BIOS's, consider the case of Bob Wallace. Bob's the author of PCWRITE (a word processor). "Compiled" PCWRITE is available "free" from any number of SIG distributors; yet he has sold over 2,000 copies of his source code to registered owners. (I paid my $75.) As he says, who is going to rip him off on the compiled version price? That is, how can you beat the SIG price: free, except for a scratch disk and postage. I have half a dozen word processors, either as a consequence of writing reviews or by earlier foolish purchase.

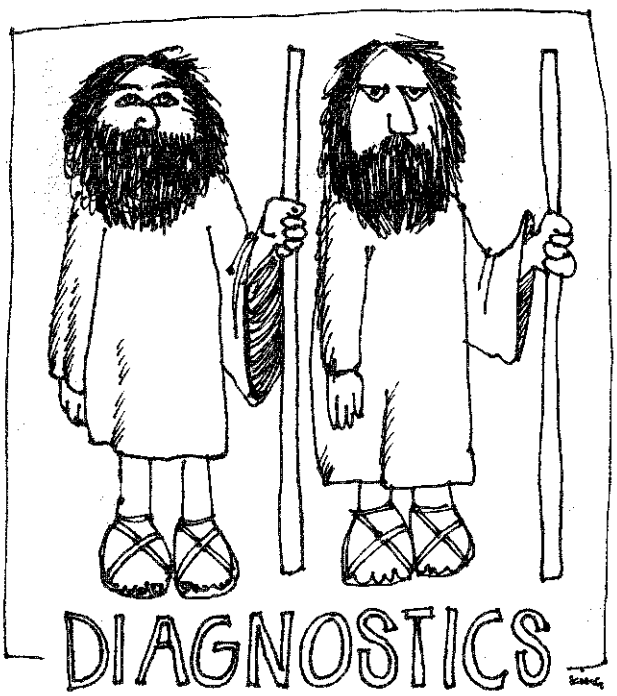
I bought PCWRITE because the Pascal source code allows me to integrate it into my MS-DOS environment. I can make any changes I want. I'm in charge. I suppose the new Volkswriter Deluxe fresh out of the box is a better word processor, but Lifetree doesn't sell their Pascal source and that means I am limited to the features they've packaged into it. Not so with PCWRITE, and 2,000 times $75 in six months is not shabby.

CONCLUSION/ADVICE

Copy protect your software during development and when you are showing it around. It is possible to come up with simple yet elegant protection schemes that stop the automatic copy programs. Most of the "suits" who will use their automatic programs to rip you off are not hackers and if you stop their automatics, you've stopped them for good. They'll not pay the couple thousand nor take the considerable legal risk to hire a copier-hacker at that stage (unless you are a big-time writer, in which case there is nothing this article teaches you that you don't already know). Don't think a "logical" protection scheme will be easy.

Don't expect to easily find or transform yourself into that mythical adolescent who is able to stop COPYII PC or COPYWRITE. I don't know a "copier" under 40 years of age or less than 140 IQ with fewer than twenty years of coding experience. Frankly, outside of the movies, these old lions eat the kids' lunch (real world).

When you go to market, the numbers change. If you elect copy protection, you've got to go to laser-burns and trick BIOSs to stop the best of the copiers. Once your product starts to sell big, the copiers must try to break your protection. If only to defend their claims that they can "copy everything." At the earlier stages (when you don't even have a distributor) you weren't a target, which is why trivial protection might work. I can stop those programs, but I'm sure the




copiers could take my scheme apart in a day once they looked at it with Mechanic or Explorer. So my recommendation is that you make sure your protection works before you let people see it and even then — if you made the disk on an IBM PC — accept the fact that it can be broken by the pros the same day it falls into their hands. McQuaid, a copy-software writer, requests that his clients forward any diskette that his COPYWRITE package can't copy, and he gives them a reward. The only problem is that few of his clients are willing to forward the original (why?) and he can't learn what the problem is from anything less than the original. The COPYWRITE produced version that won't run teaches him nothing — think about it. Common sense should tell you that your "logical" protection won't be as good as the ones from VISI, CONTEXT, LOTUS, that have been broken in the past. That's why the ballgame, at the end-user level, has changed to trick BIOSs and laser-burns.

*Richard Larratt*
*Rural Route #1*
*Demorestville, Ontario*
*Canada, KOK 1W0*
(613) 476-5309

COPY UTILITY PROGRAMS

Central Point
Software, Inc.
P.O. Box 19730-203
Portland, OR 97219
(503) 244-5782
(COPYII PC)

MLI Microsystems
P. O. Box 825
Framingham, MA 01701
(617) 926-2055
(DISK MECHANIC)



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Quaid Software Limited
620 Jarvis Street, Suite 2412
Toronto, Ontario
CANADA
(416) 961-8243
(COPYWRITE)

HARD DISK PROTECTORS

Award Software, Inc.
236 North Santa Cruz Ave.
Los Gatos, CA 95030
(408) 395-2773
(Win-Lock)

Vault Corporation
1320 S. Baldwin
Arcadia, CA 91006
(213) 514-8594
(Prolok)

LOGICAL COPY PROTECTION

Olive Branch Software
1715 Olive Street
Santa Barbara, CA 93101
(805) 569-1682
(SLK/F)

SoftDesign
6 Blodgett Rd.
Lexington, MA 02173
(617) 861-0902
(SoftDesign Protect)

Micro-Software Developers, Inc.
214 1/2 West Main Street
St. Charles, IL 60174
(312) 377-5151
(custom copy protection)

HACKER PRODUCTS

Morgan Computing Co., Inc.
10400 N. Central Expwy., Suite 210
Dallas, TX 75231
(214) 739-5895
(full-line, including TRACE 86)


THE MOST EXTENSIVE C LIBRARY ANYWHERE

THE GREENLEAF FUNCTIONS
Library for C Programmers
Total Access to IBM PC and PC XT
Fully Compatible with CI C86
Lattice and Microsoft C,
DOS 2.0 and DOS 1.1

A GOOD WAY TO LEARN C

FULL SOURCE CODE

PARTIAL CONTENTS

◆ DOS 2.0 – over 25 functions ◆ Complete Video Access for Text and Graphics ◆ Over 60 String Functions (including "justify" and "center") ◆ Rainbow Series Color Text ◆ Time and Date ◆ Over 40 IBM Graphics Printer functions ◆ Total Access to Function and Special Keys ◆ RS232 Async ◆ All BIOS Functions ◆ Some functions are "BASIC Equivalent" ◆ Software Diagnostics ◆ Disk functions ◆ Utility functions ◆ and more . . .

THE GREENLEAF FUNCTIONS . . .
Nearly 200 functions, 180 page manual, 3 Libraries, Extensive Examples of each function, Full Source Code in C and Assembler, Demo Programs

Save Months of Painstaking Work . . .

$175.00
Add $7.00 for postage and handling. Specify Compiler – (C86, Lattice, or Microsoft)
MasterCard and VISA Accepted
Price subject to change without notice.
Dealer Inquiries Welcome
(214) 446-8641

FOR THE IBM PC AND PC XT

The Greenleaf Functions is a trademark of Greenleaf Software. C86 is a trademark of Computer Innovations Inc. IBM, IBM PC, PC XT, and PCDOS are trademarks of International Business Machines Corp. Lattice is a trademark of Lattice Inc. Microsoft C and MSDOS are trademarks of Microsoft Inc.

GREENLEAF SOFTWARE ◆ 2101 HICKORY DRIVE ◆ CARROLLTON, TEXAS 75006


Norell Data Systems
3400 Wilshire Boulevard
P.O. Box 70127
Los Angeles, CA 90010
(213) 257-2026
(DISK MAGIC, BACKUP)

Visual Age
642 N. Larchmont Blvd.
Los Angeles, CA 90004
(213) 439-2414
(Code-Smith, a disassembler) □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## 8088 ASSEMBLY MACROS

*by Ron Burk*

I constructed the accompanying macros which mechanize some of the repetitiveness of 8088 programming. They are used to create, enter and return from functions or procedures. The BP register is maintained in such a way that arguments to functions and local variables are located at fixed offsets from it.

The five macros are much easier to use than they were to make. The BENTRY macro defines the entry point of a function. You supply it with the name of the function and the names of the arguments to the function (which should have been pushed on the stack by the caller). It saves the current BP register on the stack and changes it to point to the top of the stack. It also defines the names of the arguments to the function as positive offsets to BP, like [BP+2], [BP+4], etc.

The AUTO macro defines "automatic" storage for local variables to the function. These variables are offsets in the stack and automatically disappear when the function returns. The SP register is decremented in order to allocate the named variables, which are really negative offsets to the BP registers (for example [BP-2]).

The SAVE macro saves the named registers on the stack in a certain order. It also sets flags so that the BEND macro knows what (if any) registers have to be popped.

The BEND macro defines the end of the function and pops off any saved registers in the correct order, then readjusts the stack. It also defines a label (consisting of the function name followed by _end) so that you can easily jump to the return code.

Finally, the BCALL macro pushes arguments (if any) on the stack and then calls the named functions.

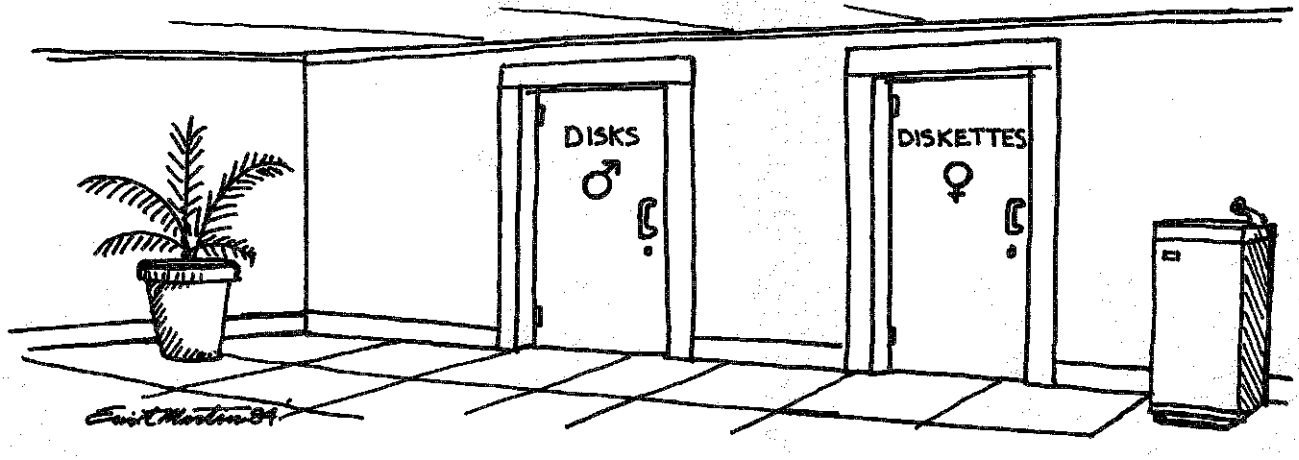
The macros include some rudimentary error checking in order to catch obvious mistakes in using them. They also print function names at the beginning and end of each function, which sometimes makes it easier to find mistakes. The following skeleton shows prototypical use of macros:

```
bentry  myfunc,<arg1,arg2,arg3>
        auto    <local1,local2,local3>
        save    <cx,dx,flags>
        (body of function)
bend    myfunc
```

In this example, the macros would cause the symbol arg1 to be equal to [BP+8]. The BEND macro expands into code to restore the saved registers and adjust the stack, then execute the proper RET instruction. It also defines the label myfunc_end to be used as a target for branches to the exit point from within the function.

It is hard to construct a realistic example that is reasonably small, so I have included a dummy example of a MAIN program that can be assembled to demonstrate the expansion of each macro. Although my assembler doesn't seem to mind, the assembler documentation states that it is an error to redefine a symbol via EQU. To avoid doing this, the macros employ some trickery. If a local variable named FRED is to be defined, it is defined with an EQU to be [BP+FRED@], and FRED@ is defined (or redefined) to be the correct decimal offset via the "=" operator.

Even for people who dislike one or more aspects of the macros, they are useful in a strictly didactic sense, for they provide examples of many features of the MS-DOS assembler's macro facility. These macros are free to anyone who is willing to type them in.






PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        page    ,132
;
;       MAIN PROGRAM, showing use of macros
;
;
        .xlist
include bmacs.asm
        .list
;
pgmseg  segment
        assume  cs:pgmseg,ss:pgmseg,ds:pgmseg,es:pgmseg
main_entry:
        mov     bp,sp           ; Initialize BP
        bentry  main            ; Define main function
        auto    <len>
        nop                     ; Body of function 'main'
        cld                     ; goes here
        cld                     ; (whatever)
        bcall   fun1,<len>      ; Might call something else
        nop
        bend    main
;;;;
        bentry  fun1,<len>      ; Ok to use 'len' again
        auto    <a,b,c>         ; Local variables
        save    <flags,cx,es,bx>        ; Save registers if you want
        nop
        nop
        cld                     ; Body of function 'fun1'
        je      fun1 end        ; Label of return of 'fun1'


        nop
        bend    fun1


pgmseg  ends
        end     main_entry

;-----------------------------------------------------------------
;;
;;    BMACS.ASM  - Macros for 8088 Assembly Language
;;
;;
;;;;
```

```
@argdef    macro    argnames    ;; Internal mac for defining args
@nargs=0                        ;; @nargs is # of bytes for arguments
ifnb<argnames>                  ;; If there were any arguments
    irp    dummy,<argnames>     ;; First, count # of args
    @nargs=@nargs+2             ;; 2 bytes per argument
    endm
    @i=@nargs+2                 ;; Allow 2 bytes for return address
    irp    arg,<argnames>       ;; For each argument
    arg&&@=@i                   ;; Define a redefinable offset
    arg equ [bp +arg&&@]        ;; Define name itself
    @i=@i-2                     ;; Next arg is 2 bytes lower
    endm
endif
    endm                        ;; End @argdef macro


;;;;
@err    macro    msg            ;; Print error message if in pass 1
if1
    %out msg
endif
    endm                        ;; End of @err macro


;;;;
@check    macro    extra,macname    ;; Checks for bad invocation
ifnb<extra>
   @err <You forgot angle brackets in macname macro>
endif
    endm                        ;; End macro @check


;;;;
bentry    macro    func,argnames,mistake    ;; Defines procedure
@nlv=0                  ;; @nlv is # of bytes needed for local variables
@nsr=0                  ;; @nsr is # of bytes needed for saved regs
@savefl=0               ;; Flag that says the save macro has been used
irp    flag,<@ax,@bx,@cx,@dx,@si,@di,@ds,@ss,@es,@flags>
    flag=0              ;; Initialize register flags
endm
ifb<func>
    @err <Missing function name on bentry macro>
else
    @err <!>!>!>Begin func>
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
endif
@check    mistake,bentry
@argdef <argnames>              ;; Define argument offsets
func    proc                    ;; Define a procedure
    push    bp                  ;; Save previous BP
    mov     bp,sp               ;; Make new BP
    endm                        ;; End bentry macro

;;;;
auto    macro    vars,mistake   ;; Defines local variables
if @savefl                      ;; Must not use save before auto!
   @err <***ERROR*** must not save before auto>
endif
@check mistake,auto
irp var,<vars>
    var&&@=@nlv+2               ;; Add 2 to skip old BP
    var equ [bp - var&&@]
    @nlv=@nlv+2
endm
    sub     sp,@nlv             ;; bump SP past local vars
    endm                        ;; End auto macro

;;;;
@pop    macro    reg            ;; Internal macro to pop a register
if @&reg                        ;; If we saved this reg
    ifidn<reg>,<flags>
    popf                        ;; Use correct inst for flags
    else
    pop     reg                 ;; Or ordinary pop
    endif
endif
    endm                        ;; End @pop macro

;;;;
bend    macro    func           ;; Define end of procedure
ifb<func>
    @err <Missing name on bend macro>
endif
@err <!<!<!<End func>
func&_end:                      ;; Define an ending label
irp  reg,<flags,es,ss,ds,di,si,dx,cx,bx,ax>
    @pop reg
endm
if  @nlv                        ;; If any local vars
    add sp,@nlv                 ;; Release local vars
endif
    pop     bp                  ;; Restore old BP
    ret     @nargs              ;; Return and pop args
func    endp                    ;; Define end of procedure
    endm


;;;;
@save    macro    r1,r2         ;; Famulus of save macro
ifidn<r1>,<r2>                  ;; If they are the same
    @&r1=1                      ;; Set flag variable
    @found=1                    ;; Tell caller we found it
endif
    endm                        ;; End @save macro


;;;;
save    macro    regs,mistake   ;; Macro for saving registers
@savefl=1                       ;; For error checking
@check    mistake,save
irp  reg,<regs>
    @found=0                    ;; Haven't found reg yet
    irp  regname,<ax,bx,cx,dx,si,di,ds,ss,es,flags>
    @save  regname,reg          ;; Set corresponding macro variable
    endm
    ife     @found              ;; If we don't know that name
    @err <***ERROR*** 'reg': Unknown register name in save>
    endif
endm
irp  reg,<ax,bx,cx,dx,si,di,ds,ss,es,flags>
    if @&&reg                   ;; If we're supposed to save that one
    ifidn<flags>,<reg>
        pushf
    else
        push  reg
    endif
    endif
endm
    endm                        ;; End save macro


;;;;
bcall    macro    func,args,mistake    ;; Macro for calling procedure
@check    mistake,func
ifnb<args>                      ;; If we're supposed to push anything
    irp arg,<args>
    push    arg                 ;; Push whatever on the stack
    endm
endif
    call    func
    endm                        ;; End of bcall macro □
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

# THE PROGRAMMER'S SHOP

helps compare, evaluate, find products. Straight answers for serious programmers.

**SERVICES**

- **Programmer's Referral List**
- **Compare Products**
- **Help find a Publisher**
- **Evaluation Literature free**
- **BULLETIN BOARD - 7 PM to 7 AM 617-461-9174**
- **Dealer's Inquire**
- **Newsletter**
- **Rush Order**
- **Over 300 products**

**Our Free Report: PRODUCTIVITY - MSDOS**

Assume use of compiler and typical editor. What commercial or public domain products, what techniques improve productivity? "Productivity with MSDOS" is a growing document with some answers. Call to request it. **Help improve it. Earn $50** credit toward any purchase when we add any description, code, or idea received from you.

**RECENT DISCOVERIES**

PROFILER - Examine MSDOS program execution speeds. Determine where to improve programs in any Microsoft language, Lattice, or C86. Make histograms that show time spent in portions of your program, and doing MSDOS I/O, etc. $175.

**"C" LANGUAGE**

| | | LIST PRICE | OUR PRICE |
|---|---|---|---|
| APPLE: AZTEC C - Full, ASM | | $199 | call |
| 8080: BDS C - Fast, popular | | 150 | 125 |
| 8080: AZTEC C - Full | | 199 | call |
| Z80: ECOSOFT - Fast, Full | | 250 | 225 |
| 8086: C86 - optimizer, Meg | | 395 | call |
| 8086: Lattice - New 1.1 & 2.0 | | 500 | call |
| Microsoft (Lattice) | MSDOS | 500 | call |
| Digital Research - Megabyte | 8086 | 350 | 269 |
| Desmet by CWare - Fast | 8086 | 109 | 99 |

**BASIC**

| | ENVIRONMENT | LIST PRICE | OUR PRICE |
|---|---|---|---|
| Active Trace - debug | 8080/86 | $ 80 | 72 |
| MBASIC-80 - MicroSoft | 8080 | 375 | 255 |
| BASCOM-86 - MicroSoft | 8086 | 395 | 279 |
| CB-86 - DRI | CPM86 | 600 | 419 |
| Prof. BASIC | PCDOS | 345 | 325 |
| BASIC Dev't System | PCDOS | 79 | 72 |

**EDITORS Programming**

| | | LIST PRICE | OUR PRICE |
|---|---|---|---|
| C Screen with source | 8080/86 | NA | 60 |
| EDIX - clean | PCDOS | 195 | 139 |
| FINAL WORD - for manuals | 8080/86 | 300 | 215 |
| MINCE - like EMACS | CPM, PCDOS | 175 | 149 |
| PMATE - powerful | CPM | 195 | 175 |
| | 8086 | 225 | 195 |
| VEDIT - full, liked | CPM, PCDOS | 150 | 119 |
| | 8086 | 200 | 159 |

**FORTRAN**

| | | LIST PRICE | OUR PRICE |
|---|---|---|---|
| MS FORTRAN-86 - Meg | MSDOS | $350 | $255 |
| SS FORTRAN - 86 | CPM-86 | 425 | 345 |
| FORTRAN-80 - 66 decent | CPM-80 | 500 | 350 |
| INTEL FORTRAN - 86 | IBM PC | NA | 1400 |
| DR FORTRAN COMING | | | |
| RM FORTRAN COMING | | | |

**LANGUAGE LIBRARIES**

| | | LIST PRICE | OUR PRICE |
|---|---|---|---|
| C to dBASE interface | 8080/85 | $150 | $140 |
| C Tools 1 - String, Screen | PCDOS | NA | 115 |
| C Tools 2 - OS Interface | PCDOS | NA | 92 |
| FLOAT 87 - Lattice, PL1 | PCDOS | NA | 115 |
| GRAPHICS: GSX - 80 | CPM80 | NA | 75 |
| HALO - fast, full | PCDOS | 200 | 175 |
| Greenleaf for C - full, 200+ | PCDOS | NA | 165 |
| ISAM: Access Manager - 86 | 8086 | 400 | 300 |
| BTRIEVE - many languages | PCDOS | 245 | 215 |
| PHACT - with C | PCDOS | NA | 250 |
| FABS | CPM80 | 150 | 135 |
| PASCAL TOOLS - Blaise | PCDOS | NA | 115 |
| SCREEN: Display Mgr. 86 | 8086 | 500 | 375 |
| PANEL-86 - many languages | PCDOS | 295 | 245 |
| **WINDOWS for C** | PCDOS | NA | 115 |
| Virtual Screen - Amber | PCDOS | 295 | call |

**PASCAL**

| | ENVIRONMENT | LIST PRICE | OUR PRICE |
|---|---|---|---|
| PASCAL MT + 86 | CPM86/PC | $400 | $279 |
| without SPP | CPM80 | 350 | 239 |
| MS PASCAL 86 | MSDOS | 350 | 249 |
| SBB PASCAL-great, fast | PCDOS | 350 | 315 |
| PASCAL 64 - nearly full | COM 64 | 99 | 89 |
| **SBB Jr - best to learn** | PCDOS | NA | 95 |

**OTHER PRODUCTS**

| | | LIST PRICE | OUR PRICE |
|---|---|---|---|
| Assembler & Tools - DRI | 8086 | 200 | 159 |
| COBOL - Level II | 8086 | 1600 | 1275 |
| CODESMITH-86 - debug | PCDOS | 149 | 139 |
| GC LISP | PCDOS | 495 | 475 |
| IQ LISP - full 1000K RAM | PCDOS | 175 | call |
| Janus ADA - solid value | PCDOS | 500 | 449 |
| **MBP Cobol-86 - fast** | 8086 | 750 | 695 |
| Microshell improve CPM | 8080 | 150 | 125 |
| Microsoft MASM-86 | MSDOS | 100 | 85 |
| PL/1-86 | 8086 | 750 | 530 |
| PLINK-86 - overlays | 8086 | 350 | 315 |
| POWER - recover files | 8080/86 | 169 | 139 |
| READ CPM86 from PCDOS | PCDOS | NA | 55 |
| READ PCDOS on an IBM PC | CPM86 | NA | 55 |
| Trace 86 | PCDOS | 125 | 115 |

**FEATURES**

C INTERPRETERS for MSDOS - Ask about one for beginners for $85 or full development for $500.

C HELPER includes source in C for MSDOS or CPM80 for a DIFF, GREP, Flowcharter, C Beautifier and others. Manage your source code easier. $125.

PROLOG86 Interpreter for MSDOS includes tutorials, reference and good examples. Learn in first few hours. For Prototyping, Natural Language or AI. $125.

**Call for a catalog, literature, and answers**

**800-421-8006**

**THE PROGRAMMER'S SHOP™**

128-J Rockland Street, Hanover, MA 02339.
Visa 617-826-7531, Mass: 800-442-8070 MasterCard

Note: All prices subject to change without notice. Mention this ad. Some prices are specials.

Ask about COD and POs.
All formats available.

# PROLOG-86™

## Learn Fast, Experiment

1 or 2 pages of PROLOG would require 10 or 15 pages in "C."

Be familiar in **one evening.** In a **few days** enhance artificial intelligence programs included like:

- **an Expert System**
- **Natural Language** (generates dBASE display)

Intro Price: $125 for PCDOS, CPM-86.
**Full Refund** if not satisfied.

## CONTEST: "Artificial Intelligence Concepts"

$1,000 Prize, Recognition for applications in PROLOG-86™ that teach, are clear, illustrate. Call for details. Deadline 11/31/84.

**SOLUTION SYSTEMS™**
45-J Accord Park, Norwell, MA 02061
**617-871-5435**

# C HELPER™

## UNIX™-like Utilities for C Programming with source

**Save time** when working with your C programs. Full source lets you make them work your way, helps you learn.

Utilities included: compare files (DIFF), cross reference your variables (CCREF), examine the flow of functions as they call each other (FCHART), format and indent programs (pretty printer), search for patterns (GREP). Others check program syntax, print programs your way and more.

UNIX is a trademark of Bell Labs.

$135 for PCDOS, MSDOS, CPM-86 or CPM80.

Call with questions or for "Programming with C Helper".

**SOLUTION SYSTEMS™**
45-J Accord Park, Norwell, MA 02061
**617-871-5435**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## RANDOM ACCESS FILES IN BASIC

*by Ronald Moak*

Microsoft's BASIC handles two types of files, random and sequential. Each type of file structure has benefits for the programmer. In this article we will cover the statements and functions used to create and work with random access files. A follow-up article will use the concepts covered here to create a simple database management program.

In general, writing code to use random access files is the same for both interpretive and compiled BASIC. However, there is a difference in the number of files you may have active, and the maximum record size. Throughout the article I will indicate possible differences between the two.

Random access files are particularly useful for holding information that must be processed rapidly, because you can pull out records at any point in the file rather than searching sequentially starting with the first record. They are also distinguishable from sequential files in a number of other ways. Foremost among the differences is the ability to read or write data from any place in the file, without the need to read or write all of the preceding records. However, as is often true in life, to get something you must give up something in return. With random files, all the records must be the same length, even if the information you wish to store doesn't take up that much space.

In order for your program to find a record in a random access file, its location must be calculated. The position of the desired record is found by multiplying the record length times the record number. Consequently, all records must have fixed lengths, which is determined when the file is created.

Random files allow numbers to be stored in binary format. In sequential files, all numbers are stored in the ASCII sequential file. For example, the integers 12333 can be stored in a two-byte binary number instead of the five-byte ASCII characters. Files that contain many numbers will tend to occupy less disk space than files stored in sequential format. If, on the other hand, the record regularly contains several spaces, the file would consume more disk space than a sequential file.

To move data into or out of a random access file from BASIC, the file must first be opened using Microsoft's OPEN statement. When the OPEN statement is executed, several operations are performed. First, the directory is checked to determine if the file specified in the statement exists. If the file is not found, then a file is created and added to the directory. If the file exists, a buffer is assigned to it. A buffer is an area in memory where data is stored prior to being written to or read from a file.

Syntax for the OPEN statement may take one of two forms listed below.

OPEN filename AS filenumber LEN = record length

or

OPEN R, filenumber, filename, record length

In both statements, the filename is a standard MS-DOS filename of eight characters with a three-character extension. The OPEN statement defaults to the currently logged drive if none is indicated. If you are logged into drive A and wish to



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

open a file on drive B, place the drive letter followed by a colon "B:" in front of the filename. For flexibility, both the filename and the drive specification can be concatenated into a variable and the variable placed in the OPEN statement. For example:

```
10 FILENAME$ = "B:" + "NAME.DAT"
20 RECORD_LENGTH% = 128
30 FILENUM% = 1
40 OPEN FILENAME$ AS FILENUM% LEN =
    RECORD_LENGTH%.
```

The next important part of the OPEN statement is the filenumber. The filenumber is the number of the buffer that is assigned to the file when the file is opened. If you need multiple files open simultaneously, each file needs to be assigned to a separate buffer. The filenumber is used in all other statements that deal with files. It is used to indicate which buffer you are working with when moving data in and out of a file.

Interpretive BASIC allows three files to be opened at a time. If more than three files are needed, BASIC must be invoked with "/F:Numfiles". "Numfiles" is the number of files needed, up to the maximum of 15 files. Programs written for the BASIC compiler may have up to 127 files open at a time.

The final part of the OPEN statement is the record length. The record length indicates the length of the record in bytes. This number is necessary for setting up the buffer length and for calculating the position of the record in a random file. If record length is omitted, the default is 128 bytes.

In interpretive BASIC, to lengthen the record beyond 128, BASIC must be invoked with "/S:size", where size is the length in bytes up to a maximum of 32767 bytes. Compiler BASIC has no restriction on the length of records.

As with the filename, the values for both filenumber and record length may be placed in variables. The variables can then be placed in the OPEN statement for greatest flexibility in generating random access files on the run.

Once we've a opened a random file and assigned it a buffer, we need a way to define the buffer so we can move data in and out. Suppose the record we want to read or write is composed of several different variable types that include both string and numeric values. To accomplish this we use the FIELD statement for mapping the contents of the buffer into pseudo variables, which will be referred to in this article as field identifiers (FI). Field identifiers are similar to variables in that they may be named like variables; however, as we shall see later, the two are not the same.

The syntax of the FIELD statement is as follows:

FIELD filenumber, width AS FI [,width AS FI]...

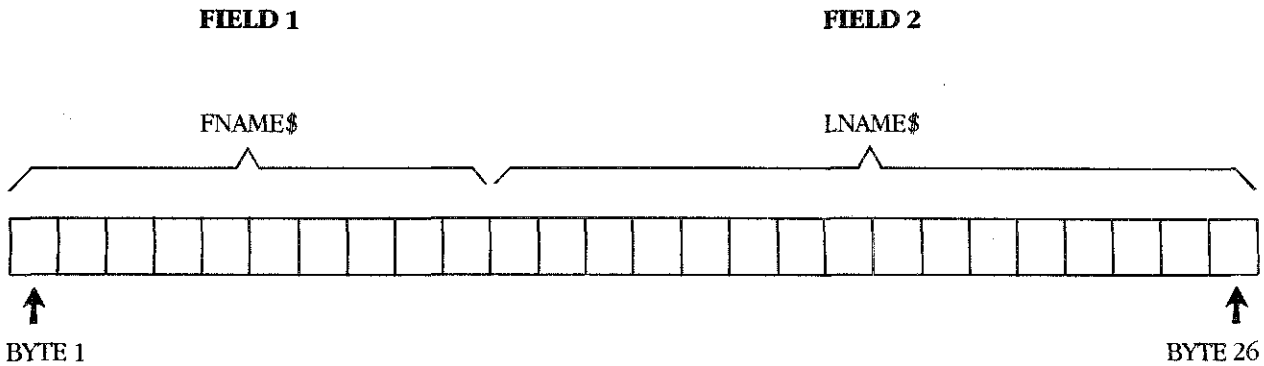
The filenumber is the same as we used in the OPEN statement.

Records may be composed of one or more fields. The width number or variable in the FIELD statement refers to the length of the field (in bytes). If the record is composed of multiple fields, a separate width and field identifier is needed for each field. The sum of all the individual field widths should equal the length of the record.

When the FIELD statement is used, the program assigns an identifier to remember how much space each variable takes up in the buffer. For example, if we issue the following FIELD statement:

```
30 FIELD 1, 15 AS LNAME$, 10 AS FNAME$, 25 AS STRT$, 2 AS
    STAT$, 5 AS ZIP$
```

The first 15 characters of the buffer would be assigned to LNAME$, the next 10 bytes would be assigned to FNAME$, the following 25 bytes would be assigned to STRT$, and so on until the end of the record. (See illustration below.)






PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

One of the powerful features of the FIELD statement is the ability to map the same buffer with different groups of field identifiers. This allows you to give different names to the same piece of information. This allows data to be moved in and out of the buffer in a way more convenient for the programmer. The buffer mapped in the FIELD statement above could also be mapped, at the same time, in the following ways:

```
40 FIELD 1, 25 AS NAME$, 25 AS STRT$, 2 AS STAT$, 5 AS ZIP$
50 FIELD 1, 25 AS NAME$, 32 AS ADDRESS$
```

As a consequence, you can transfer data in either formats. This ability is especially handy when sorting files. The entire record has a single name.

Up until now we have been using the FIELD statement for defining areas in the buffer that will hold strings. Things get more complicated when we want to place numbers into the buffer. In general, numbers in random access files are held in binary format. Therefore, we must define enough bytes to hold the number in its binary format. Different numeric variable types require different amounts of storage. Internally, integers take up two bytes of memory in BASIC, single precision numbers take up four bytes and double precision numbers take up eight bytes. The same is required for storing numbers in random access files in a binary format. If we are going to place integers in the buffer, we must allocate a minimum of two bytes to hold an integer.

```
60 FIELD 2, 2 AS NUM1%, 4 AS NUM2!, 8 AS NUM3#
```

The FIELD statement in line 60 has mapped the first two bytes in the buffer to hold an integer, the next four bytes to hold a single precision number, and the final eight bytes to hold a double precision number.

Care must be taken when describing multiple maps for a buffer that stores numeric values (binary). It is possible to map a field that overlaps both a string and numeric space. If this occurs, part of a binary number winds up in a string variable, creating garbage.

A word or two of warning concerning field identifiers. Earlier, I defined field identifiers as pseudo-variables, because they have some of the characteristics of variables. The areas in memory referenced by variables are in different locations than the file buffer referenced by field identifiers. To distinguish between field identifiers and true variables, BASIC puts some restrictions on the use of field identifiers. The primary restriction being field identifiers are limited to the right side of BASIC assignment statements.

```
40 FIELD 1, 25 AS NAME$, 25 AS STRT$, 2 AS STAT$, 5 AS ZIP$
50 LET STRINGNAME$ = NAME$      ' LEGAL

60 LET NAME$ = STRINGNAME$      ' ILLEGAL
```

If the statement in line 60 above were executed, a new variable called NAME$ would be created in the variable storage area of memory. In any future reference to NAME$, BASIC would look to the variable storage area of memory for the value. This would destroy the pointer to the data in buffer area 1 (file #1), referenced by the field identifier NAME$. As a result, any future attempt to read and write data from that field in the record would fail.

Now that we have opened the file and defined the buffer, we need a method to move data in and out of the buffer. If you paid close attention to the description of the FIELD statement, you should know how to pull string information out of the buffer. To retrieve strings, simply assign it to a variable using an assignment statement with the field identifier on the right side of the equal sign.

```
40 FIELD 1, 25 AS NAME$, 25 AS STRT$, 2 AS STAT$,
    5 AS ZIP$
50 STRINGNAME$ = NAME$
```

Because of the inability to place field identifier names on the left side of the equal sign, placing strings into the buffer is not as easy as removing them. To accomplish this task, BASIC provides the two statements LSET and RSET. When used in the fashion,

```
60 LSET NAME$ = STRINGNAME$
```

the information in the variable STRINGNAME$ is copied into the buffer area defined by NAME$. If the number of characters in STRINGNAME$ is greater than the number of characters in the field referenced by NAME$, the extra characters are truncated on the right. If the number of characters in STRINGNAME$ is less than the number of characters field referenced by NAME$, the buffer is filled in from the left, and the extra characters on the right are filled in with blanks.

The RSET statement is similar to the LSET statement, except that it will right justify the string in the buffer and pad the left blanks with spaces.

Movement of numeric data in and out of the buffer is slightly more complex than movement of string data. The binary configuration in which numbers are stored internally in BASIC is different from the way in which numbers are stored



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

in random access files. To convert numbers from internal format to file format, BASIC uses the Make-String functions. There are three functions, labeled MKI$ (for integer), MKS$ (for single precision), and MKD$ (for double precision). These are designed to convert BASIC's internal format into two, four, or eight byte strings.These strings are then placed into the file buffer the same way strings are stored using the LSET and RSET statements.

To place any number in the buffer, convert the number to the appropriate length string using the Make-String function, then place the string into the buffer using either the LSET or the RSET statement.

```
20 OPEN FILENAME$ AS 1 LEN = RECORD_LENGTH%.
30 FIELD 1, 2 AS NUM1$, 4 AS NUM2$, 8 AS NUM3$
40 NUM1% = 345
50 NUM2! = 345.67
60 NUM3# = 345.6789012#
70 LSET NUM1$ = MKI$(NUM1%)
80 LSET NUM2$ = MKS$(NUM2!)
90 LSET NUM3$ = MKD$(NUM3#)
```

Three other functions are needed to complete the movement of numeric data in and out of the file buffer. As you have probably already guessed, these functions are the complement of the Make-String functions. The Convert-String to numeric functions (CVI, CVS, CVD) converts the numeric strings to their appropriate precision. As with the Make-String functions, the last character of the function name indicates the precision of the converted number.

```
100 NUM4% = CVI(NUM1$)      'Convert string to integer
110 NUM5! = CVS(NUM2$)      'Convert string to single precision
120 NUM6# = CVD(NUM3$)      'Convert string to double precision
```

Lines 100-120 reconvert the numbers that were placed in the buffer in lines 70-90.

Now that we have a way to move data in and out of the file buffer, let's see how to move data in and out of the file. BASIC provides two statements for this purpose. The GET statement brings data from the file into the buffer. The PUT statement performs the opposite task by placing the contents of the buffer in a file. Both statements have the same parameters.

```
PUT filenumber, recordnumber
GET filenumber, recordnumber
```

The filenumber is already familiar to us. The recordnumber points to the record in a file that we want to read into the buffer or write to disk. If we want to read the tenth record into the file assigned to the buffer, we might use the following statements:

```
100 RECNUM% = 10
110 GET 1, RECNUM%
```

Using recordnumber in either the GET or the PUT statement is optional. If the number is omitted, the first record after the last record accessed will be written or read from the file.

Another word of caution: when BASIC reads a random access file, it may pull in more than one record at a time, depending upon the record length. The main benefit of this is that you do not have to access the disk every time you read or write a record. If the record is already in memory, the program will use it. Otherwise it will pull the record off the disk. However, there is a potential problem with this method. When a program that is constantly accessing random files is aborted abnormally (during a power failure, for instance), records that are still in the buffer will be lost. This version of BASIC does not provide an efficient method of flushing the buffer when a record is written to force it to disk. If you are in interpretive BASIC and abort the program using Ctrl-Break, you may save the buffer by typing CLOSE to close all open files.

This leads us to the final statement needed for using random access files. When we are done working with a file, we use the CLOSE statement. This frees the buffer and writes any remaining records to disk. The CLOSE statement may be used by itself to close all open files, or it may be followed by one or more filenumbers to close several files.

```
130 CLOSE 1,3
```

Line 130 closes the files referenced by numbers one and three. If there is an open file referenced by number two, it is left open.

At first glance, setting up and using random access files may seem overly complex and frustrating. Yet when the commands are broken down into logical steps — opening the file, defining the buffer, moving data into and out of the buffer, moving a record to or from the disk, and finally, closing the file — some of the frustration should subside.

In the next article we will use the commands and functions discussed here to create a simple database management program. □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## C UTILITY ROUTINES

*by Warren Hillman*

There are several excellent utility routine packages designed to help C programmers. Besides the usual OBJ libraries that come with the package, you normally receive the source code as well. What a deal, to say the least. Of all the packages out, the award for sheer volume must go to Novum Organum, author of C Building Blocks. They sell over 600 routines in convenient function sets: string handling, file management, CRT control, keyboard, mathematics, database, communications, advanced routines, graphics and window management. They also sell a series of Hayes modem control routines.

The following source code is an example of one of the routines included in C Building Blocks. It is a pathname-filename utility. You pass the routine the address (pointer to) a string containing both a pathname and a filename. The routine separates the two, cutting off the end of the pathname with a NULL and returning a pointer to the beginning of the filename.

For more information on the extensive set of C Building Block routines, contact:

*Novum Organum*
*29 Egerton Road*
*Arlington, MA 02174*
*(617) 641-1650*

```
/************************
 * (c) 1984 Novum Organum
 *      29 Egerton Rd.
 *      Arlington, Ma. 02174
 ************************/
/* #include "basic.rpl" , not needed */

#define EOS '±0'          /*  End Of String, NULL */

/* The following routine separates the pathname or disk
specifier from its filename, and returns it in a separate
array. We do this by initially copying the combined name
into another array, then searching for a colon or slash
that ends the pathname. So one array ends up holding the
pathname with a NULL stuffed in there to end the
string, and the other array has its starting point
(pointer) moved to show where the filename begins.

Only a pointer to the filename array is returned since you
already know the address of the pathname array. */

char *StripPath (out,in)
    char    *out;
    char    *in;

    char    *fil_nam, *endpath;

    strcpy (out,in);        /* duplicate, for working, */
    fil_nam = in;           /* pointer to start of name, */
    endpath = out;          /* and end-of-path pointer to stuff NULL */


    while
        (*in)

            /* look for slashes or colons, */
        if    (*in == ':')
                                /* if a colon, */
            endpath= ++out;     /* point to name, */
            fil_nam = ++in;

        else
        if    (*in == '/')
                                    /* if we find a slash, */
            endpath= ++out;          /* move the pointer up, */
            fil_nam = ++in;

        else
        if    (*in == '±±')

            endpath= ++out;     /* keep moving it up, */
            fil_nam = ++in;

        else

            ++in;
            ++out;

    *endpath = EOS;         /* delete the name part, */
    return(fil_nam);    /* return ptr to name part. */
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## USING THE PMATE MACRO LANGUAGE

*by Robert E. Brown, Landmark Software*

Anyone attempting to document a product as rich in features as PMATE, a text editor for the IBM PC, is clearly in a no-win situation. Inexperienced computer users are bound to complain about the amount of material to be digested. Highly experienced computer users are bound to complain about inadequate detail. And almost everyone in between will wish for more examples illustrating how the features can actually be used. So it is virtually inevitable that the PMATE Users Manual doesn't fully meet the needs of all users.

The purpose of this article is to help new PMATE users who have some prior programming experience; we'll assume that you have a copy of the PMATE Users Manual and you've read through it at least once. We'll discuss those items that we wish had been more fully explained when we first started learning to use the PMATE macro language. Experienced PMATE users will find most of this old hat — but we think that this information will enable the relatively new user to start writing macros much faster.

### MAKING VARIABLES LOCAL

Because macros can be called from within other macros — like subroutines — it's important to be able to make variables local so that they can be used freely within a macro without affecting variables with the same name in the calling macro. The use of local variables is not restricted to subroutine macros. In XERGO, a PMATE-based word processing environment which I am marketing, a keystroke macro may be executed while a prompt from some other macro is pending. This requires that variables used by the keystroke macro be local so that there won't be interference with other active macros to which control will eventually return.

Although variables in the macro language are inherently global, they can easily be made local by using the numeric stack. Suppose you want to make the three variables @6, @7, and @8 local. Just start the macro like this:

```
@6,@7,@8, ; Push the values of @6, @7, and @8 onto the stack —
          ; we want to use @6, @7, and @8 as local variables.
```

You can then freely use these variables without risking any interference with other active macros, provided you terminate the macro with this code:

```
@SV8@SV7 ; Restore original values to @6, @7 and @8.
@SV6     ;
```

Note that the restoration must be in reverse order (the original value of @8 is on the top of the stack because it was pushed last, thus @8 must be restored first) because "@S" always refers to the top of the numeric stack. Remember that any reference to @S pops the stack, i.e., removes the top item from the stack. So after @SV8 is executed, the top of the stack will be the original value of @7.

One major caveat: Many macros make provision for an abnormal or early exit — to permit the user to abort at a prompt, for example. When using the numeric stack to create local variables, it's imperative that the original values get restored prior to the logical end of the macro on all possible paths through the macro. So everywhere you would otherwise use a "%" (exit macro), you will instead use a branch command, say 'JE', where SECTION E is typically at the physical end of the macro:

```
:E       ; SECTION E: Perform end-of-macro processing.
@SV8@SV7 ; Restore original values to @6, @7 and @8.
@SV6     ;
%        ; Exit macro.
         ; END SECTION E.
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The "%" command is required if SECTION E isn't at the physical end of the macro; the "%" command is optional if SECTION E is at the physical end of the macro. But if you subsequently add another section after SECTION E, you'll have to remember to put in the "%" command. So you may prefer to routinely include the "%" command even when it's not required — it's only one extra byte.

## DETECTING THE END OF THE LINE

Because of auto-disk-buffering, it's possible to edit files much larger than the available memory. If you use the "Z" command to move the cursor to the end of the buffer, you'll find (by using @T) that the cursor is on a null (ASCII 0). Similarly, if you use "U" to move the cursor to the end of the file, you'll also find that the cursor is on a null. So we need a way to distinguish "end of file" from a mere "end of buffer."

There is one special case where this is very easy: if a null is reached as the result of an "L" command, then we've reached the end of the file. The "L" command will bring additional text into the buffer, if necessary, in order to place the cursor at the start of the next line, i.e., "L" will force auto-disk-buffering to occur if necessary. So, assuming that the file doesn't in fact contain any nulls, an "L" command will terminate on a null only when reaching the end of the file. Even when the end-of-buffer null is in column 0, an "L" command from the preceding line will cause more text to be brought into the buffer so that the "L" command will terminate on the true first character of the next line rather than on the end-of-buffer null. So, in those cases where an "L" command is being used anyway, the end-of-file test is very simple:

```
L         ; Move cursor to start of next line; this forces
          ; auto-disk-buffering if necessary.
@T=0JE    ; Branch to end-of-macro processing (SECTION E)
          ; if cursor is on a null.
```

Often, however, a null is reached in some other manner. For example, if you're examining each successive character and encounter a null, you'll need to determine whether you reached the end of the file. Here's an algorithm that works in this more general case:

```
@T=0[     ; If cursor is on a null, then
          ; SECTION: Determine if you're at end of file.
@X,@L,    ; Push current column number and current
          ; line number onto the stack.
L         ; Attempt to move cursor to start of next line:
          ; If cursor is merely at end of buffer, this
          ; will force auto-disk-buffering to occur;
          ; but if cursor is at end of file, nothing
          ; will happen.
@S-@LL;   Restore cursor to original line and column;
@SQX      ; if auto-disk-buffering occurred, the
          ; character at that location will no longer
          ; be a null; if no auto-disk-buffering
          ; occurred, the cursor will still be on
          ; the null.
          ; END SECTION: Here you're at end of file if and
          ; only if cursor is still on a null.
@T=0JE    ; Branch to end-of-macro processing (SECTION E)
          ; if you're at end of file.
]         ; End if.
```

Say the null is in column 25 of line 500. If that's the end of the file, the "L" will do nothing; so (@S−@L)L is 0L and @SQX is 25QX; thus the cursor will still be on the null. But if the null is merely the end of the buffer, "L" forces more text to be read in and puts the cursor at the start of line 501; then (@S−@L)L is −L and @SQX is 25QX. The cursor is restored to column 25 of line 500, but the character there is no longer a null but rather the next character of the file.

Note that under this scheme "@L" is referred to only in the relatively infrequent case where the cursor is on a null. In the typical case where a loop is being executed, speed is increased considerably because a reference to "@L" can take as long as .25 second if the cursor is near the end of the buffer. So it's imperative that "@L" not be referenced on each pass through a loop unless absolutely necessary.

## TYPES OF STACKS

The macro language utilizes four stacks: the numeric stack, the iteration stack, the macro stack, and the computation stack. The numeric stack is manipulated via the "," push operator and the "@S" reference which pops the stack. The other three stacks are used by the macro language processor itself. The iteration stack is used to keep track of the nesting of "if" and "loop" structures made with brackets or braces. The macro stack is used to keep track of the nesting of macros. The computation stack is used in deciphering arithmetic expressions that use parentheses. If the capacity of any of these stacks is exceeded, a STACK FULL error occurs. You need to know about the various stacks so that you will understand which stack is overflowing when a STACK FULL occurs.

The capacity, or maximum size, of these four stacks varies in different versions of the macro language. Although you don't normally have to worry about this, it makes an interesting exercise to check out the maximums in your particular version. You can use the following small macros — one at a time in, say, buffer 1 — to determine the maximum size of each stack for future reference. We've added some extraneous blanks to make it easier to decipher the commands:

Numeric stack: Put "0V4 [1, VA4]" in buffer 1 and execute the command ".1". The value of @4 following the STACK FULL is the maximum size — 75 in PMATE/XERGO 3.36.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Iteration stack: Put "0V4 :A [VA4 JA" in buffer 1 and execute the command ".1". The value of @4 following the STACK FULL is the maximum size — 30 in PMATE/XERGO 3.36.

Macro stack: Put "VA4 .1" in buffer 1 and execute "0V4.1". The value of @4 following the STACK FULL is the maximum size — 30 in PMATE/XERGO 3.36.

Computation stack: One method is to put "?(((((((((((((((((((((((((((((((" in buffer 1 and then execute the command ".1". Just count the number of times you can press, say, "±" before you get a STACK FULL — 15 in PMATE/XERGO 3.36.

If it isn't obvious why these procedures work, take the time to figure them out, referring to the PMATE Users Manual when necessary. You may find it helpful to put a "?" in front of the commands placed in buffer 1 — that will let you step through the execution one command at a time.

MOVING THE CURSOR TO THE END OF THE LINE

If, on your own, you can write a macro that will always move the cursor to the end of the current line, you've probably already gained considerable experience writing macros. Suffice it to say at the outset that the obvious method doesn't always work. Lest we scare you away with an extended discussion of such a seemingly simple topic, we hasten to add that you don't normally have to worry about this. You'll normally have a macro that performs this function for you. But attempting to create such a macro brings to light many insights that you'll use over and over again.

Let's begin by asking what we mean by the end of the current line. Except for the last line of the file — as viewed from within PMATE/XERGO — each line ends with a carriage return character (ASCII 13), so the column containing the CR constitutes the end of the line. The last line of the file ends with a null character (ASCII 0); so the column containing the null is the end of the line in that case. Often the null is preceded by a Control Z (ASCII 26). This is basically a superfluous carryover from the days of CP/M — you shouldn't assume that a Control Z will be there when using PC DOS or MS DOS.

The "L" command moves the cursor to the beginning of the next line — provided there is a next line. If the cursor is already on the last line of the file, "L" will simply move the cursor to the end-of-file null.

Suppose the cursor isn't on the last line of the file. Then "L" moves the cursor to column 0 of the next line. If you think of the file as a sequential list of characters, the carriage return at the end of the original line is the character immediately preceding the character at the start of the next line. Thus the command "L-M" moves the cursor to the end of the current line - provided it isn't on the last line of the file.

Unfortunately, "L−M" doesn't work if the cursor is on the last line of the file in this case. 'L' puts the cursor on the null at the end of the line, and "−M" puts it on the character preceding the null. If we knew in advance that the cursor was on the final line of the file, we'd just use "L" to move the cursor to the end-of-file null. Thus it seems natural to modify the algorithm this way: First use "L"; if the cursor is then on a null, you're done; otherwise use "−M". So the proposed solution is "L@T=0'[−M]." This seems so obviously correct that it may come as a surprise that it isn't.

Consider the case where the current line ends with a CR but the very next character is the end-of-file null. The proposed solution puts the cursor on the null on the next line instead of on the CR at the end of the current line. It's worth a moment's reflection to see where we got off track. We know "L−M" works, provided we're not on the last line. "L@T=0'[−M]" works fine if we're on the last line, but doesn't necessarily work if we're not on the last line. We assumed that if "L" put the cursor on a null, then we were on the last line to begin with. Alas, that assumption is incorrect.

Exercise: Why doesn't "L-M@T=13'[M]" work? Hint: If the cursor isn't on the last line of the file, it does work. So the failure, if any, must occur in some situation where the cursor is on the last line of the file.

Here's an algorithm that does work:

```
@T=0'[      ;If cursor is not on a null then
L           ; advance cursor to character after the next CR
            ; or to the null if there are no more CR's.
-M          ; Move cursor back one character — to the CR
            ; unless there are no more CR's.
@T=13'[     ; If cursor is not now on a CR then
M           ; Advance cursor one character to the null.
]           ; End if.
%           ; Exit macro — speeds up execution.
][          ;Else (cursor is on a null).
@L,         ; Push current line number onto stack for
            ; reference below.
L           ; Move cursor to start of next line if possible.
@S=@L'[     ; If cursor isn't still on the original line then
            ; Here auto-disk-buffering has brought more
            ; text into memory.
-M          ; Move cursor back to the CR at end of original
            ; line.
]           ; End if.
            ; Note that if '@S=@L', it implies that the
            ; cursor was on the last line of the file to
            ; begin with and is now on the null.
]           ;End if-then-else.
```

Admittedly, this is not a pretty solution, but it does work and it's reasonably efficient because the references to @L occur only in the unusual case where the cursor is on a null. If you



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

can find a more elegant, more efficient, or shorter method, please share your discovery.

One more point needs to be mentioned. Suppose the cursor is on a CR. Does that imply that it's at the end of the line? No, for when the cursor is moved to the right of the actual location of the end-of-line CR, it will still test as though it were on the CR. The intervening "empty space" doesn't get blank-filled until a character is inserted there. When we execute a command to move the cursor to the end of the line, we want the cursor to go to the true CR location, even if the cursor is to the right to begin with. Thus, even if we know the cursor was on a CR, we have to "L-M" to move the cursor to the end of the line. Similarly, if the cursor is to the right of the end-of-file null, it still tests as though it is on the null. Executing an "L" command in this situation moves the cursor to the left, to the true location of the null. With these characteristics in mind, reread the macro code above to convince yourself that it will work properly even if the cursor is initially beyond the true end of the line.

Now that we have an algorithm that works, what do we do with it? If, as in the case of XERGO, the same algorithm is available in the supplied macro library, just use the facility provided. Otherwise, it's a simple matter to select an unused single-character name and add the macro to the library. In either case, bear in mind that it's necessary to use this algorithm only when there's the possibility that the cursor may be on the last line of the file. In all other cases it's more efficient to use "L-M", which executes faster and avoids the overhead of locating the macro within the macro library. It's worth mentioning that with a 20K macro library, it takes anywhere from 0 to .25 second to begin executing a macro, depending on whether it's at the beginning or end of the library. Consequently, the most frequently used macros, particularly those that might reasonably be used in loops, should be placed near the beginning of the library.

## PRESERVING THE CURSOR LOCATE BETWEEN BUFFERS

Most macros use at least one buffer in addition to the buffer containing the text being processed. Sometimes we need to pick up a piece of information from another buffer or build a command in another buffer. Whatever the specifics may be, we generally need to switch to another buffer and then eventually return to the original buffer. When we return, we'd like everything to be the same as when we left — particularly the cursor location and the lateral screen alignment.

Let's look at the techniques for returning to the original buffer. The system variable "@B" contains the buffer number of the current buffer, not to be confused with the name of the current buffer. For example, the buffer number of buffer 2 happens to be 3. In practice, this causes very little difficulty.

Beginning with version 3.24 of the macro language, it became possible to use a variable containing the buffer number instead of the buffer name in buffer commands. That is, if has the value 3, then 'BE' will switch to buffer 2, just like "B2E". Similarly, if the top of the numeric stack is 3, "B@SE" will switch to buffer 2, beginning with version 3.31. Only user variables and @S may be used in this way; other expressions may not be used. This is not really a problem, because any desired expression can be pushed onto the stack and we can then use "B@SE".

So there are two easy techniques for returning to the original buffer: store the original buffer number in a variable or on the numeric stack. Note that the latter option is not available if, while in the other buffer, you'll need something you've previously pushed onto the stack. And remember that if you use the first method, you should usually make the variable local at the start of the macro.

Suppose we switch to another buffer for some reason and then eventually return to the original buffer. Assuming we haven't explicitly moved it, will the cursor in the original buffer still be at the original location? At least through version 3.36 of the macro language, the answer is "not necessarily." If the cursor was beyond the end of the line, it will have reverted to the end of the line. You'll want to work around this quirk in most of your macros. (By the time you read this, it's possible that a newer version of the macro language will fully preserve the cursor location automatically.)

The first, and most general technique, assumes that you have two variables available, say "@7" and "@8", that have been made local and aren't being used for any conflicting purpose. Before changing buffers, use this code:

```
@XV7    ; Store current column number in @7 for eventual
        ; restoration.
BV8     ; Store current buffer number in @8 for eventual
        ; return.
```

Then you can switch to some other buffer. After you're done there, execute this:

```
B@8E    ; Return to original buffer.
@7QX    ; Restore cursor to original column.
```

Note that this technique doesn't use the numeric stack.

If you're won't need to refer to anything previously placed on the numeric stack while you're off in some other buffer, you can use a simpler technique instead. Before switching buffers, do this:

```
@X,     ; Push current column number onto the stack.
@B,     ; Push current buffer number onto the stack.
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

To return after you're done in the other buffer, do this:

```
B@SE    ; Return to the original buffer.
@SQX    ; Restore cursor to original column.
```

Remember that while you're in the other buffer, you don't have access to anything previously pushed onto the stack using this technique.

These two methods make it easy to return to the original buffer with the cursor still at the original location.

## LOCATION TAGGING

Many commands in the macro language can take a numeric argument specifying the number of lines or number of characters that are to be affected. For example, "nD" deletes n characters starting at the cursor. In such cases, a variable or expression may be used as the numeric argument. For instance, if @5 is 10, the @5D will delete the next 10 characters. In practice, however, it's relatively unusual to use a variable or expression as the numeric argument. Reason: It's usually easier to use "taggging."

"Tagging" is a method of specifying a block of contiguous characters of text. The tagging consists of two parts. First, you tag a specific location by executing the tag command "T" — this tags the cursor location. Then you move the cursor and refer to the resulting block by using "#" instead of a numeric argument.

This tagging process involves two cursor locations: the location of the cursor when you execute "T" and the location of the cursor when you use "#" instead of a numeric argument. The "T" location may be physically earlier or later than the "#" location. But exactly what block are we referring to when "#" is used? The block extends from the earlier physical location within the file up to but not including the later location. For example, suppose the text contains "ABC" and you execute "T" while the cursor is on "A". If you then move the cursor to "B" and use "#" instead of a numeric argument, the block consists of just the letter "A" — it doesn't include "B". Exactly the same result occurs if you use "T" while the cursor is on "B" and then move the cursor to "A" before using "#". This implies that the character at the initial tag location may or may not be part of the block — it depends where the cursor is when you use "#".

There's one more question you might ask about what we've said so far: What happens if the "T" location and the "#" location are the same? Answer: The block is empty.

One of the most interesting aspects of the tagging process is that it can even be used with commands that normally work only on complete lines. For example, "nBbD" copies n lines, starting at the cursor, from the current buffer into buffer "b". But if we tag a location using "T", move the cursor to another location, and then execute "#BbD", the specified block of text will be copied — even though the block doesn't necessarily end at the end of a line. This makes it easy to copy individual words or phrases.

## AN EXAMPLE OF TAGGING

To illustrate one more use of tagging, suppose you wanted to develop a macro that would delete the current line, including the CR, if any, and leave cursor at the start of the next line, if any. (Actually, such a macro is usually provided for you, but it makes an interesting example.) Recall that "0L" moves the cursor to the start of the current line, and "K" deletes text from the cursor location up to and including the CR at the end of the line. So "0LK" would seem to do the trick — but it doesn't.

For suppose the cursor is on the last line of the buffer but not the last line of the file. The "K" command will only delete up to the null, indicating the end of the buffer. This produces two problems: First, only part of the line will be deleted — or possibly none of it! Second, the cursor won't move to the beginning of the next line.

The solution is to make sure the entire line is in the buffer before you attempt to delete it. Once again we make use of the fact that the "L" command will force auto-disk-buffering to occur, if needed:

```
0L      ; Move cursor to start of current line.
T       ; Tag the first character of the line.
L       ; Move cursor to the start of the next line,
        ; if possible; this causes more text to be read
        ; in, if necessary; if cursor is on last line of
        ; file, leave cursor on end-of-file null at the
        ; end of that line.
#D      ; Delete from start of original line through the
        ; character preceding the cursor, leaving cursor
        ; on the character it was on before executing '#D'.
```

Note that if the original line is not the last line of the file and hence ends with a CR, we delete from the start of that line through the CR, leaving the cursor in column 0 of the next line. But if the original line is the last line of the file and thus ends with a null, everything is deleted from the start of the line up to but not including the null. The cursor will be left on the null, which will now be in column 0 We can't go to the next line in this case because there is no next line.

## SEARCH FACILITIES

The macro language provides the ability to search either forward or reverse, to restrict the search to specific lines, to search the text currently in the buffer, or to search through



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the entire file. To make effective use of these facilities, we need to know the answers to questions like these:

Where does the search begin relative to the initial cursor location?

Where is the cursor left following a successful search?

Where is the cursor left following an unsuccessful search?

Searching in the forward direction begins with the character at the cursor. If a forward search is successful, the cursor will be positioned after the final character of the string found.

Searching in the reverse direction begins with the character preceding the cursor. If a reverse search is successful, the cursor will be positioned on the first character of the string found.

Because of these differences, you'll sometimes find it more convenient to search in the reverse direction. For example, suppose you were changing every character with an ASCII number less than 32 to a space (ASCII 32). You might use the wildcard "control W" to search for "word terminators." A word terminator is anything other than a letter or digit. Therefore, every control character is a word terminator, but not every word terminator is a control character. Consequently, after locating a word terminator, you need to test to see if it is in fact a control character. This is easier when searching from right to left because the cursor will be on the character found, and you won't have to reposition the cursor in order to perform the test.

Prior to executing any search command from within a macro, you should almost always use the "E" command to suppress the error message if no such string exists. You can then use the system variable @E to determine if the search was successful: @E will be true if and only if the search failed. The value of @E reverts to 0 (false—search succeeded) after each reference. So you can't test it twice unless you store it somewhere. Generally, there's no reason to want to test it more than once.

In order to use searching successfully within a macro, it's imperative that you keep in mind where the cursor will be if the search fails. This in turn depends on what type of search command you're using.

An "S" or "-S" command, where the number of lines is not specified, will always leave the cursor at the original location if the string can't be found. Note that the search terminates in this case when the end or beginning of the buffer is reached.

An "nS" or "-nS" command, where n specifies the number of lines, will generally leave the cursor at the point of failure. For example, an "-0S" command, which searches from left of the cursor to the start of the same line, leaves the cursor in column 0 if it fails. A "1S" command, which searches forward from the cursor through the CR at end of the line, leaves the cursor in column 0 of the next line if it fails. In other words, the cursor is left in the same location as if the search had succeeded on the last possible character. There is one major exception: if during the course of the search the end of the buffer is reached, the search will terminate with the cursor still at its original location.

An "nUS" or "-nUS" command, where n specifies the number of lines, will always leave the cursor at the point of failure when it aborts. The same is true for "US" and "-US", which search forward or reverse until the end of the file is encountered.

To summarize this situation, all variations that include "US" will leave the cursor at the point of failure. "S" and "-S", provided the number of lines is not specified, will leave the cursor at the original location if it fails. "nS" and "-nS", where n is specified, will leave the cursor at the point of failure unless the end of the buffer has been reached. Don't feel bad if you have to keep looking this up.

One caveat when using a search string that will match a carriage return (ASCII 13): if you're searching in the forward direction using "1S" or "1US", the string won't match the CR at the end of the line. For example, suppose you're using the search string consisting of the "control N" wild card plus a space. This string should match any nonblank character, so it should certainly match a CR. But "1S" or "1US" would fail if the first nonblank was the CR at the end of the line: @E would test true (search unsuccessful), and the cursor would be left in column 0 of the next line. This is the kind of quirk that might well disappear in a subsequent version of the macro language, so we advise against relying on it. Meanwhile, try to avoid using "1S" or "1US" when working with a search string that will match a carriage return. If possible, use "S", "US", "2S", or "2US" instead, or search in the reverse direction.

## CAVEATS WHEN USING THE NUMERIC STACK

The vast majority of macros do not on balance add or remove any items from the numeric stack. That is, whatever items a macro may push onto the stack are popped before it terminates. In this typical situation, you must make sure that this occurs on every possible path through the macro. It's very easy to neglect to balance the stack on infrequently traveled paths — for example, when the macro terminates abnormally, either because the macro recognizes that successful completion is impossible or because the operator chooses to abort at a prompt.

Some macros are intended to add one or more items to the numeric stack. Most commonly, one item is added to the stack on balance as a flag, typically to indicate whether the



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

macro terminated normally or abnormally. Here, too, it's imperative that the same number of items be added to the stack on balance on all possible paths.

On rare occasions a macro may be written to place a variable number of net items on the stack, with the top item indicating the number of net additions.

In order to make it a bit easier to detect a stack imbalance, one of the XERGO macros that is used at the end of most major macros sounds the bell three times if it detects a numeric stack imbalance. This works even if the macro is supposed to add net items to the stack. It's a good exercise to ponder how this might be accomplished. The value lies in trying various paths, including the rarely traveled ones.

AVOIDING COMMON MISTAKES

Remember that the macro language uses reverse polish notation — so each operator acts on the result achieved thus far, unless you use parentheses to alter the order of evaluation. Thus, when testing whether a variable equals a negative constant, you should place the minus sign at the left or you should use parentheses. For example, suppose you want to test whether @6 equals -1. The boolean expression should be written as "-1=@6" or "@6=(-1)". Suppose you instead write the test as "@6=-1" and suppose that @6 equals 4. You want the value of the boolean expression to be 0 (false), but instead it will be 3. This will cause the commands in the body of the "if" to be executed three times instead of not being executed at all. When the boolean operators "and" (&) and "or" (!) are used, the expression following the operator must be put in parentheses unless it happens to be a single variable:

```
@K>47& ;
(@K<58)[ ; If the operator pressed a digit then
```

Without the parentheses, the reverse polish notation would produce an unwanted result, as in the preceding example. When using the "nVm" or "nVAm" commands, add an ESC character (ASCII 27) to the command if "m" is a single digit and the very next command begins with a digit. This is necessary because the latest versions of the macro language now permit two-digit variable names. Thus "0V82VA7" would be interpreted as "0V82 VA7", whereas you probably intended "0V8 2VA7".

Remember that an "Else" or "Else if" clause always starts with "][" and that no character is permitted between the two brackets.

When a "loop" structure without an iteration count follows an "if" structure, put an ESC character (ASCII 27) in front of the opening bracket of the "loop" structure.

For example:

```
                ;   .
                ;   .
                ;   .
]               ; End if.
$[              ; START LOOP: ...
                ;   .
                ;   .
                ;   .
```

Without the ESC character (which we've printed as a dollar sign), the loop would be interpreted as the "else" clause in the "if" structure. Because these situations come and go during the early stages of developing a macro, I suggest you not worry about them until you're ready to start testing. At that point it's very easy to find these situations by scanning the structured pseudo-code for "START LOOP": just make sure the first preceding nonblank character of the macro code isn't a closing bracket. During subsequent modifications, do be on the lookout for these cases. If you prefer, you might wish to include the ESC routinely on any loop that doesn't have an iteration count.

When popping the numeric stack with @S and discarding the result, make sure that the "@S" isn't followed by a command that takes an optional boolean argument or an iteration count. For example, suppose you want to pop the stack, discard the result, and then terminate the macro:

```
@S      ; Pop the stack, discarding the result.
1%      ; Exit macro.
```

Here the "1" (or some nonzero integer) is required. Otherwise you'd have '@S%', which would not execute the exit command if the top of the stack happened to be zero. It wouldn't suffice to add an ESC after "@S".

Remember that an expression at the end of the body of a loop will be interpreted as a termination test. For example, suppose you want to discard the top of the stack and continue looping (presumably there's an EXIT LOOP statement somewhere in the body of the loop):

```
[       ; START LOOP: Each pass ...
        ;   .
        ;   .
@S      ; Discard top of stack.
0]      ; REPEAT LOOP (until EXIT LOOP is executed)
```

Without the "0", the loop would terminate if the top of the stack happened to be a non-zero value.

When using "case" structures — an "if" with one or more "else if" clauses — remember that there will be more than one closing bracket at the end of the case structure. There



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

will be one extra bracket for each "else if". The total number of closing brackets at the "end case" will equal 1 plus the number of "else if" clauses. Don't count an "else" unless it's an "else if." And only count those "else if" clauses that are at the same level as the initial "if." (This is explained more cogently in our earlier article "How to Use Structured Pseudo Code to Write Macros for PMATE and XERGO".)

## USING INTERACTIVE DEBUGGING

One of the features that makes programming in the macro language such a pleasure is its easy-to-use debugging facilities. Simply insert a "?" where you want to start debugging. At that point the macro will pause and permit you to step through it one command at a time by pressing, say, the reverse slash. Expression values will automatically be displayed on the status line after "ARG=" as you do this. Consequently, you see not just what commands are being executed, but also what values are resulting from the calculations — without any extra effort. Pressing the ESC key will resume normal execution until the next '?' is reached. In a few minutes at most, you'll see where the execution is going awry and why.

In most languages, the debugging facilities are relatively cumbersome, making it worthwhile to spend some time bench checking the logic whenever a problem arises. But with the macro language, it's best to use the debugging facilities right away, whenever the source of the problem is not immediately apparent. You'll find the cause of the problem much faster this way.

## CONCLUSION

In this article we've tried to provide some insight into some of the most frequently used features of the macro language. Now that you have a more complete understanding of these core features, you should feel more confident about starting to write your own macros. You'll find that you'll soon be writing macros to perform very complex tasks.

*© Copyright 1984 LANDMARK SOFTWARE — All Rights Reserved. Permission to republish this article will generally be granted upon written request, but LANDMARK SOFTWARE reserves the right to continue to use this material for promotional or other purposes.* □

**"BASIC_C" LIBRARY— BRIDGING THE BASIC GAP—**

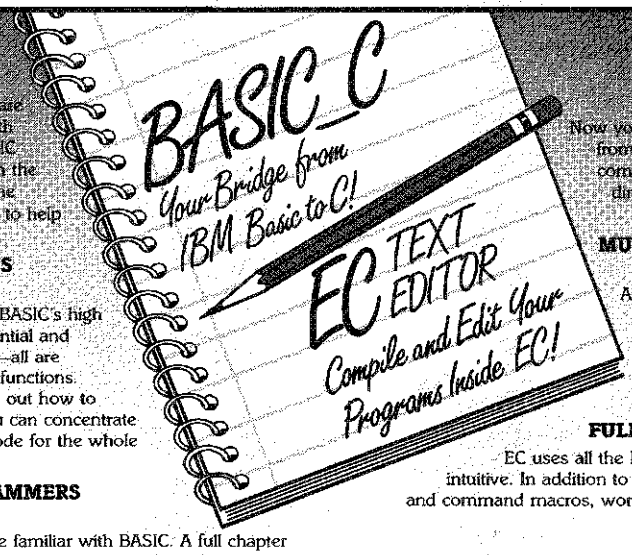
Every day, programmers like yourself are making the jump from BASIC to C. With high level functions that work like BASIC and documentation that is written from the perspective of a BASIC programmer, the BASIC_C library provides the best path to help make your journey pleasant and fast!

**BASIC'S POWERFUL FUNCTIONS IN C PROGRAMS—**

The BASIC_C library provides most of BASIC's high level functions—string handling, sequential and random i/o, graphics and many more—all are designed to look and work like BASIC functions. Instead of wasting time trying to figure out how to clear a screen or locate the cursor, you can concentrate on learning what C is about. Source code for the whole library is included!

**WRITTEN WITH BASIC PROGRAMMERS IN MIND—**

Our documentation is oriented to those familiar with BASIC. A full chapter is devoted to a point-by-point comparison of BASIC and C. Each function's description contains at least one example written in BASIC and the equivalent C. Many customers tell us that the BASIC_C documentation is the best C tutorial they've seen.




| | |
|---|---|
| EC DEMO DISK - | $ 5 |
| EC (pronounced e-see) - full screen editor | $125 |
| BASIC_C LIBRARY - including source code | $175 |
| BASIC_C & EC | $275 |
| BASIC_C & DeSmet C compiler/w debugger | $325 |
| BASIC_C & Computer Innovations C compiler | $475 |
| BASIC_C & Lattice C compiler | $525 |
| BASIC_C & EC & DeSmet C compiler/w debugger | $425 |
| BASIC_C & EC & Computer Innovations C compiler | $575 |
| BASIC_C & EC & Lattice C compiler | $625 |

**EC SCREEN EDITOR— MAKE ANY PC-DOS CALL FROM INSIDE EC—**

Now you can edit, compile and test your program from inside the EC editor. In fact, EC gives you complete access to the operating system. Do a directory, copy or delete files, even run other programs without ever leaving EC!

**MULTIPLE WINDOWS ... WITH A FILE IN EACH—**

All windows can be shown on the screen at the same time, or the screen can be dedicated to one file, while the others are kept in the background—only a keystroke away. You can even cut and paste blocks of text between windows!

**FULL SCREEN EDITOR FOR THE IBM—**

EC uses all the PC's special keys so that editing is fast and intuitive. In addition to standard editing features, EC provides text and command macros, word wrap, paragraph reformatting, horizontal scrolling, and color video control.

**DEMO DISK IS ONLY $5—**

Call or write for our full-featured demo. You'll get a standard version of EC that can handle up to 10K worth of text files and a complete set of documentation. See for yourself how pleasant it is to use an editor that gives you multiple windows and free access to your operating system.

*BASIC_C and EC run on an IBM PC, or look-alike, with at least 128K RAM under DOS 1.1 or higher. To use the EC EDITOR'S DOS interface feature you must have DOS 2.0 or higher and enough RAM to run the additional program.

IBM is a trademark of International Business Machines.
MO. Residents Add 6% Sales Tax.

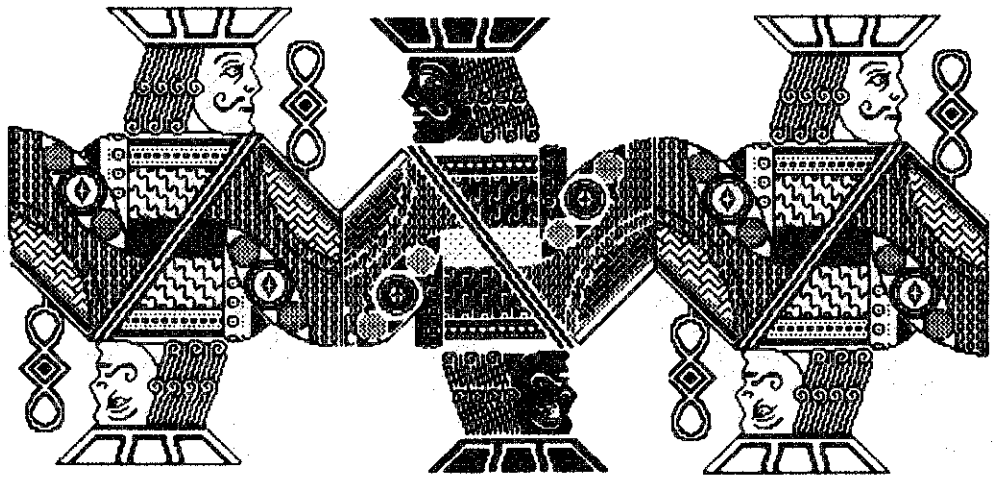
SOURCE

**C SOURCE**
12801 Frost Road
Kansas City, MO 64138

**816-353-8808**



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



KMAN dBASEII R:BASE

what database manager should you really buy ?

**dBASE II/KNOWLEDGEMAN/R:BASE**

*by Dr. John W. Gowens*

*Alice laughed. "There's no use trying," she said, "One can't believe impossible things."*

*"I daresay you haven't had much practice," said the Queen. "When I was your age, I always did it for half an hour a day. Why, sometimes I've believed in as many as six impossible things before breakfast."*

—Lewis Carroll in *Through the Looking Glass*

Every database manager on the market would have you believe the impossible—the package is all things to all people. Advertising hype is getting worse than the political hype. The first principle of selecting any software package, especially database managers, is that no package can be everything. By necessity, the package must be a series of compromises. To believe otherwise is to believe the impossible. While the Queen of Hearts in the above quote may have been successful at this, software users must remain skeptical.

Don't try to believe that a perfect database manager exists—believe that there some good ones (some better than others in the context of a specific application). The purpose of this article is to provide a series of tests that can serve as a guide to selecting a software package based on some objective measures as well as a guide to evaluating some subjective approaches. The methodology will be applied to three specific packages: dBase II, distributed by Ashton-Tate; Knowledgeman, distributed by Micro Database Systems; and R:Base, distributed by MicroRIM.

Before purchasing any package, you should evaluate it based upon a number of criteria. Only then can you procure software that fills the bill. Matching the software to the intended application is probably the single most important point when trying to satisfy a user. The definition, then, of what is "best" must include the specification of the problem before a choice can be made.

EVALUATION

Evaluation of software is more art than science. Perhaps it is the blackest of black magic in the computer domain. An ad hoc inquiry will not suffice. Even though some of the effectiveness measures mentioned here are subjective, at least they serve to provoke thought on topics not previously considered.

Evaluation in this report will focus on the following areas: documentation, database definition, modification, manipulation, file-to-file data exchanges, report generation, application development capability, sorting and indexing, error handling, performance, and an overall summary rating. The measures used for each topic will be assigned a numeric weight to assess its relative importance. This weighting func-



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

tion is a measure of the perceived importance of the attribute as it relates to the final data management solution. The actual weights assigned in this article are the author's perception of their importance. The numeric weights would be changed if we were considering a different applications environment.

Each package will be scored on a scale of 0 to 5. For each of the measures, 0 is "poor" and 5 is "excellent." A certain amount of precision will be considered; i.e. the score of 3.7 is an acceptable score. The values from each package will then be compared using a variance analysis technique. The hypothesis to be tested is:

H(0) No difference in packages

against the alternative:

H(1) A difference in the packages

The calculations will be shown in the results section.

The overall evaluation will be obtained by scaling the individual category scores to an index number having a mean of 50 and a standard deviation of 10, assuming a normal probability distribution. The result will be a composite number that can be used to compare other packages tested by the same procedure.

Unfortunately, this comparison is not necessarily logical unless the packages compared are of the same type, e.g., command driven or menu driven. It is difficult to compare dBase II and Powerbase because one is completely command driven and the other is completely menu driven. One must choose between them, based on the usage environment of the system. dBase II allows a programmer to create customized menu-driven systems that are just as easy to use as Powerbase. However, if you are not a programmer (or are unwilling to learn programming), the versatility of dBase II is lost for your application. For system developers, performance and in-depth documentation are usually the key factors. For casual users, ease of learning and ease of use are the key ingredients.

As you read this article, compare packages using the analysis of variance results and the overall index number; but also look at the specific items which are of more importance to you and your usage environment. A package may score favorably overall but not be appropriate for your application.

## METHODOLOGY

*Documentation:*

This evaluation area focuses on the documentation as a prime indicator of the ease of use of the package. The documentation is especially important in a command driven system. The documentation must explain the uses of all commands and give limitations of the commands. The documentation is the only place where the potential programmer can get the information needed to learn the specifics of the programming commands. With a menu-driven system, perhaps the documentation can be somewhat skimpier. The menu-driven system can substitute for documentation in some packages. This often becomes a value judgment on the part of the user.

| Weight | Measures |
|---|---|
| 5 | Quality of the index |
| 5 | Number of examples |
| 3 | Organization of the manual(s) |
| 3 | Quality of examples |
| 3 | Ease of finding information |
| 3 | Overall quality and impression of manual(s) |
| 2 | Information content of error messages |
| 1 | Description of program functions |
| 1 | Use of quality of graphics showing screen |

*Defining a database:*

The definition of a database is the activity of creating the file structure: naming the fields, defining the type of data in the field, and the width of the field.

| Weight | Measures |
|---|---|
| 3 | Create a file/database/table |
| 3 | Define attributes/fields/data item |
| 3 | Define record |
| 3 | Define a key |
| 3 | Define virtual files |
| 1 | Define virtual fields |

*Modifying the database structure:*

After the database has been created, there are often changes which must be incorporated into the database. Often the changes require the redefinition of the fields—adding fields, changing field names, etc. These measures allow you to judge the ease of making changes to a database structure with data in the database and without losing the data. An inability to make changes in the structure once information has been entered is an obvious deficiency. This is one of the most important functions of a database manager: to isolate the user viewpoint of the information from the storage of the data.

| Weight | Measures |
|---|---|
| 5 | Display file structure |
| 4 | Redefine the field length |
| 3 | Redefine a field name |
| 3 | Redefine the field type |
| 3 | Add fields to the record |
| 2 | Delete fields from the record |



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

*Manipulating the database:*

Data manipulation is the ability to enter data into the database easily. This includes editing data which already exists in the database as well as adding new records to the file. The emphasis here is primarily directed toward the interactive mode of operation of the DBMS. The ability to manipulate data under program control will be evaluated under the application development category.

| Weight | Measures |
|---|---|
| 5 | Edit records using forms format |
| 5 | Display file contents |
| 4 | Edit records using full cursor keys |
| 3 | Add new records |
| 3 | Modify (edit) records |
| 3 | Change data in fields globally/conditionally |
| 3 | Maintains index automatically for data entry |
| 3 | Sum numeric fields in file to variable |
| 2 | Delete records |
| 2 | Support picture clauses during data input |
| 2 | Buffer of key stroke input |

*File-to-file data manipulations:*

File-to-file manipulations are the type of data transfers that take information from one file and move it to another file. This corresponds to the "transaction to master" concept of file update procedures. It is an important concept for systems that require data to be entered and audited before the permanent entries are placed into the master file. Many accounting functions follow this model: payroll, time card entry, order entry, etc. Another important aspect is the desirability of making backups under program control.

| Weight | Measures |
|---|---|
| 5 | Copy file to another file, SDF* format (export data files) |
| 5 | Import "foreign" data files |
| 5 | Append records from "foreign" file |
| 4 | Copy file to another file, internal format |
| 4 | Append records file to file, internal format |
| 4 | Update records in file from another file (transaction to master concept) |
| 3 | Sum numeric fields into another file |

*SDF means standard data format.

*Report generation features:*

Report generation refers to the built-in capability of the system to prepare reports. This set of measures is not meant to evaluate reporting features that may be programmed into a system developed in the programming language of the DBMS.

| Weight | Measures |
|---|---|
| 4 | Total on specified numeric field |
| 4 | Outputs selected records in report format |
| | Output for report directed to: |
| 4 | screen |
| 4 | printer |
| 4 | file |
| 3 | Subtotal on specific numeric field to give at least one control break |
| 2 | Define and save report format |
| 2 | Redefine report format |
| 2 | Access more than one file in report format |
| 2 | Format reports to match "overlay" (e.g., Pre-printed forms) (conditional selection on logical expression) |

*Applications development features:*

Applications development features refer to the ability of the DBMS to be used as a development system to prepare menu-driven systems for use by naive computer users.

| Weight | Measures |
|---|---|
| 5 | Program system using command files |
| 5 | Supports stack level of subprograms |
| 5 | Flush edit buffer by command |
| 5 | Call host language from command files |
| 4 | Use all commands in command files |
| 4 | Supports four major programming structures (straight line, DO until, DO while, conditional branching) |
| 3 | Open multiple data files simultaneously |

*Sorting and indexing:*

Sorting and indexing demonstrate arranging the file in a particular order. Sorting the file physically orders the records of the file into the sequence specified by the sort criteria. Indexing the file does not alter the physical order of the records but rather creates an index file which maintains the order by pointing to the appropriate record. Index files are also called inverted files. Theoretically, a file may have many index files in existence simultaneously, but the file can have only a single physical order.

| Weight | Measures |
|---|---|
| 5 | Index on single key |
| 5 | Perform search for record based on key field |
| 5 | Perform search for record on non-key field |
| 3 | Index on concatenated key |
| 2 | Sort on key |

*Error handling:*

The error handling testing addresses the ability of the software to recover from user errors. Error handling is limited to the internal trapping by the software and does not reflect the programmer's ability to manually capture them.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

| Weight | Measures |
|---|---|
| 5 | Provides clear error message |
| 5 | Avoids loss of data |
| | Common errors: |
| 5 | Syntax |
| 5 | Drive door open |
| 5 | Write protected disk |
| 4 | Printer off |

*Performance:*

Performance can be measured using typical workloads based on the type of package under evaluation. Database managers obviously have different workload requirements from spreadsheets. The performance of any software package is affected by the type of equipment on which the testing is conducted. Database managers are especially sensitive to the use of a hard disk. For system developers, writing a system of any size whatsoever will require the use of a hard disk. If the DBMS is being used by a novice, the file sizes will probably be small enough that the difference between a hard disk and a floppy disk will not be too significant.

| Weight | Measures |
|---|---|
| 5 | Time to locate a record with index on |
| 5 | Time to index the file on secondary key |
| 4 | Time to relate two files to form a third |
| 4 | Time to run a report with subtotals (34 records, four columns) |
| 4 | Time to update 14 records in master file from transaction file |
| 4 | Time to add a new record to the file |
| 4 | Time to perform a global edit, changing text of one field for all 1000 records |
| 4 | Time to copy file to a backup file |
| 4 | Time to export the file as SDF |
| 3 | Disk space used to hold test file (theoretically 141KB) |
| 3 | Time to import the test file |
| 3 | Disk space to store the test file |
| 3 | Time to sort the test file on the key |
| 3 | Time to locate last record without index(worst case) |
| 2 | Time to load the DBMS(worst case) |

The database managers were evaluated using a "real" data file having 10 fields of varying field length from 5 to 30 characters. Six of the fields were character and the remaining four were numeric. Data contained in the fields were "real" in the sense that values were created for every field in the database that resembled actual data for an employee file. The database had 1000 records, each containing 141 bytes of information.

The first step in the procedure was to import an ASCII standard data file (SDF) into the database manager. This activity provided the data file created for the benchmark in the format of the individual database manager. Each of the tests defined above were conducted three times. These results were averaged to arrive at the reported times.

The testing reported here was conducted on a standard IBM PC, 384k RAM, DOS 2.0 with BUFFERS = 15 and VERIFY ON, an Epson MX-100 printer, and a Percom 10 megabyte Winchester hard disk. All testing requiring disk access was done on the Percom hard disk.

*Overall evaluation:*

The overall evaluation is the composite score formed by combining the preceding categories of evaluation into an index with a mean of 50 and a standard deviation of 10.

## FINAL RESULTS

Results of the procedure were evaluated by performing an analysis for every area. A separate score was recorded for each measure. Except for performance measures, the score was assigned on a subjective basis by the author. The package with the best/worst performance for a a specific measure was assigned the highest/lowest score. The weighted average score was then calculated using the following formula.

average score = (SIGMA) (Weight * Score) / SIGMA Weights

That is, multiply each weight by the score assigned for the measure in an evaluation area. Add the products together for each evaluation area and divide by the sum of all the weights for that evaluation area. This is called the average score for that evaluation area for that package. The package average is found by averaging the evaluation area average scores for each package separately. The grand average is obtained by averaging the three package totals.

Table 1 provides the results for the three packages evaluated for this article. As shown, each of the packages performs well in some evaluation areas but not in others. dBase II scored highest in the file-to-file manipulation, data manipulation, applications development and performance evaluation areas.

Knowledgeman scored highest in the sorting and indexing, report generation, and modify structure evaluation areas. R:Base scored highest in the documentation, report generation and define database evaluation areas. Notice that the scores are very similar in each of the evaluation areas.

Table 1
Evaluation Results
Weighted average scores for each evaluation area

| Evaluation Area | dBase II | Knowledgeman | R:Base |
|---|---|---|---|
| Documentation | 2.9 | 1.1 | 3.1 |
| Define database | 2.3 | 2.3 | 2.6 |
| Modify structure | 2.4 | 3.9 | 2.3 |
| Data manipulation | 3.8 | 3.4 | 3.2 |
| File-to-file | 3.9 | 1.7 | 1.0 |
| Report generation | 2.4 | 3.5 | 3.5 |
| Appl. development | 3.4 | 2.9 | 1.6 |
| Sorting, indexing | 3.5 | 4.7 | 2.3 |
| Error handling | 1.5 | 1.3 | 2.3 |
| Performance | 4.2 | 3.1 | 2.2 |
| Package averages | 3.03 | 2.79 | 2.41 |
| Grand Group Average | | 2.74 | |



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Table 2
Analysis of Variance (ANOVA)

| | sum of squares | df | mean squares |
|---|---|---|---|
| between groups | 1.955 | 2 | 0.9775 |
| within groups | 24.174 | 27 | 0.895 |

statistic for rejection

F(.05,2,27) = 3.35

F = 0.9775/ 0.895 = 1.09 which implies that we cannot reject.

The implication of the analysis of variance is that there is no difference in the three packages that can be decided on a statistical basis. There are differences, of course, but they are more of form than of substance. A further implication is that database managers (and probably other packages as well) must be evaluated in the context of their use. This would seem to indicate that test results currently being published in many publications should be analyzed carefully to determine the methodology and to determine if the methodology is representative of your use of the package.

## PERFORMANCE

The performance measures are extremely important to system developers. For this reason, Table 3 provides the results of the performance testing conducted in this study. Times to perform the benchmark tests are given in the following format for all tests, "minutes:seconds.tenths" except the disk storage requirements which are given in KB. The table shows that dBase II performs well in the file-to-file areas of the benchmark testing while Knowledgeman performs well in the sorting and indexing categories of the testing.

Table 3
Performance Measures

| Test | dBase II | Knowledgeman | R:Base |
|---|---|---|---|
| Time to load the DBMS | 0:1.5 | 0:2.4 | 0:5.2 |
| Time to import test file | 1:18 | 5:56 | 5:40 |
| Time to sort file on key | 8:20 | 3:24 | n/a |
| Time to index | 3:38 | 1:27 | 1:35 |
| Time to locate last record (no index) | 0:22 | 0:1.0 | n/a |
| Time to locate record with index on | 0:1.0 | 0:1.0 | 0:1.0 |
| Time to relate two files to form third (38 rec) | 1:13 | n/a | 0:46 |
| Time to run report | 1:44 | 2:53 | 2:24 |
| Time to update 14 recs | 3:40 | n/a | n/a |
| Time to add new record | 0:1.2 | 0:1.0 | 0:1.0 |
| Time to perform global edit | | | |
| a) 14 records | 0:19 | 0:44 | 0:53 |
| b) all 1000 rec | 0:41 | 0:53 | n/a |
| Time to copy file | 1:48 | n/a | n/a |
| Time to export file as SDF | 2:20 | 8:36 | 7:32 |
| Disk space (theoretical of 141 KB) | 142KB | 146KB | 167KB |

## OVERALL

The overall evaluation was computed by calculating a z-score from the standard unit normal probability distribution using the following formula:

z-score = package average − grand avg / std error

This z-score was then scaled to an index number with a mean of 50 and a standard deviation of 10 by the following formula.

index = 50 + (10 * z-score)

'The choice of 50 and 10 is arbitrary. They were selected because they will give values between 0 and 100 in every practical instance.

The overall evaluation index for the database packages is as follows:

Table 4

| package | index |
|---|---|
| dBase II | 53 |
| Knowledgeman | 51 |
| R:Base | 46 |

The index results show that there are very few real differences in the packages. The differences are in the way some features are implemented and in a few capabilities. The evidence is even stronger for making a decision about a software package on the basis of the job it must perform rather than on performance characteristics only.

## CONCLUSIONS

The command structures of the three systems tested are very similar. This is especially true for Knowledgeman and dBase II. Both systems perform virtually the same activities in database management. The key words are different of course, but the functions are very similar. One big difference in the two is that Knowledgeman has an embedded limited spreadsheet which dBase II does not have. On the other hand, dBaseII supports a host language interface and file-to-file manipulations which Knowledgeman does not support. The assembly language interface in dBase II permits access to the operating system calls; this is especially interesting in the IBM PC environment with the EXEC function which permits the running of any program in another portion of memory while keeping dBase resident. In this way, other programs can be run simultaneously with dBase II — for example, Lotus 1-2-3. R:Base does not support programming as such, but it has an interface to FORTRAN and Pascal which allows the calling of R:Base routines as subroutines. This capability could be very impor-



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

tant to you, if your applications environment uses FORTRAN or Pascal.

Knowledgeman is a superior query language when used in the interactive mode, especially when multiple tables need to be accessed in the same series of select commands. This is indicated by the high scores in the report generation area and the sorting and indexing area. Knowledgeman has no relational operators in obvious command structures. The SELECT command from Knowledgeman subsumes join and project operations to create the same effects. The strength of Knowledgeman is its ability to perform database query operations. Knowledgeman might best be used where files are interrelated only for the purpose of report output, especially if the output must be dynamically sorted at output.

Programming in Knowledgeman will probably be a little more difficult to learn for users coming from a BASIC or FORTRAN environment. Knowledgeman requires the declaration of abstract data types before any reference is made to the data type. This programming structure is more familiar to Pascal or C programmers. Unfortunately, the Knowledgeman manual does not adequately state this requirement and there are no sample programs in the manual! Knowledgeman is definitely not for the novice; however, it can be used to create menu-driven systems for a novice user.

R:Base is probably the purest relational database manager on the market. It contains every relational operator except division and the cross product. This, unfortunately, requires that the user be schooled in the use of relational operators and in relational database concepts in order to to use R:Base effectively. The strength of R:Base is its query language which adheres to the relational model and its FORTRAN and Pascal language interfaces.

R:Base would probably function best in a more scientific environment where the language is used in the interactive mode. Programming, as it is usually conceptualized, is not possible in R:Base; no looping and no conditional branching make it very difficult to write programs. R:Base can run text command files that are straight line "programs": i.e., a complicated query that is done repeatedly can be captured in a text file. Rumor is that the capability to perform looping and conditional branching will be included in the next release of R:Base. It follows that R:Base is not an ideal package for the novice user.

dBase II is more a database language than a DBMS. dBase II is no doubt the easiest of the three to learn and to use, especially in the programming mode. On the other hand, dBase II certainly is not a relational database manager in the sense of R:Base. dBase II contains no true relational operators even though it supports a limited join command. The strength of dBase II is the capability of file-to-file manipulations, which neither Knowledgeman nor R:Base have.

For system development that requires file manipulation utilizing "transaction to master file" type updates, dBase II seems to be the choice. Systems that require an audit trail are especially in mind here. Accounting systems would exemplify the type of system for which transaction file activity is important. While one could write code to perform transaction file updating in Knowledgeman, the question would have to be asked, "Why did I buy a database manager?" If I have to write code to do file updating, why not write it in COBOL and be done with it? Because dBase II seems to be the easiest to learn, the less experienced users might be inclined to learn it faster. But dBase II is not really a package for the rank beginner. The user should have some experience with files and programming before tackling dBase II; however experienced programmers can write programs that the inexperienced user will be able to use.

OTHER CONSIDERATIONS

Each package has a number of especially annoying factors with which the programmer must contend. Several of these are enumerated below.

- KNOWLEDGEMAN

  - Does not return cursor to the normal cursor when exiting the package.

  - Cannot get a directory when in Knowledgeman.

  - Cannot copy a file while in Knowledgeman.

  - Does not reset the end-of-file character when using the COMPRESS command; hence the disk space is not recovered.

  - Command syntax is never checked. You can type garbage and Knowledgeman does not know the difference.

  - Commands are verbose. There is no way to abbreviate commands. The SELECT command is especially annoying because there is no default to get all fields from a table without typing every field name in every command. If you make a typo in the command, the entire command must be retyped.

  - Apparently disk output can only be directed to the default disk using the E.ODSK and #DSKOUT variables. The commands will not allow assignment of drive names in front of file names.

- R:BASE

  - Cannot get a directory from inside R:Base.

  - Cannot copy a file while in R:Base.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

# YOUR CODE MAY BE WASTING ITS TIME! THE PROFILER™ CAN HELP . . .

- **Statistical Execution Profiler**
- **Works with any language**
- **Completely configurable**
- **Up to 16 partitions in RAM/ROM**
- **Time critical code optimization**
- **Abnormal code behavior tracking**
- **Graphic presentation of results**
- **Easy to use menu interface**

**THE PROFILER** is a software package which gives you, the programmer, a powerful tool for locating time consuming functions in your code and allows you to performance tune your program. With the **THE PROFILER** you can determine where to optimize your code for maximum benefit, then measure the results of your efforts.

Using **THE PROFILER**, you can answer questions like:
Where is my program spending its time?
Why is my program so slow? What is it doing?
Is my progam I/O bound? CPU bound? Are data buffers large enough?
How much improvement did my changes make?

**THE PROFILER** is completely software based and consists of a system resident driver and a monitor program. The memory partitions can range from 1 byte to 1 megabyte in size and can be anywhere in the address space.

**NO ADDITIONAL HARDWARE IS REQUIRED!**

Requires an IBM PC or compatible system with a minimum 64k and one drive.

**THE PROFILER** is available for $175.00 from DWB Associates or ask your software dealer. To order or for more information, call or write DWB Associates. VISA/MC accepted. Dealers welcome.

dwb Associates
P.O. Box 5777
Beaverton, Oregon 97006
(503) 629-9645

IBM is a trademark of IBM Corp. MSDOS is a trademark of Microsoft Corp.
**THE PROFILER** is a trademark of DWB Associates.

# Introducing the Creative Genius. . .

**YOU.** **Discover how easy programming can be with DataBurst.™** A unique runtime screen processor and source program generator, DataBurst™ will decrease your program development time and increase the value of your application programs. The *unique* DataBurst™ screen editor provides fast, easy screen design. Program independent screen formats reduce both development and maintenance time.

During execution of your program, DataBurst™ controls all user interaction through one assembly language interrupt service routine, requiring as little as 14K of memory. A true full-screen processor, DataBurst™ allows unlimited design complexity, and brings a mainframe advantage to your IBM® PC.

DataBurst™ is available through your local computer retailer or directly from Key Solutions, Inc. To order directly, please send check or money order for $225* to Key Solutions, Inc., P.O. Box 2297, Santa Clara, CA 95055. Additional language support (BASIC Compiler and C Compiler) is available for $40.* (Please inquire about release dates for other language interfaces).

IBM® is a registered trademark of IBM Corporation.
DataBurst™ is a trademark of Key Solutions, Inc.

*In California add applicable sales tax.

© Copyright 1984 Key Solutions, Inc.

The Design Tool for the Creative Programmer

DATA BURST™

KEY SOLUTIONS™


- When defining databases and relations, you must go all the way through without stopping (please don't get interrupted!).

- When in the edit mode, changing an attribute value causes the cursor to be returned to the beginning of the row. This can be annoying when you are changing several attributes in the same row.

- Edit mode always begins at the top of file. It does not seem to be possible to move to a preselected record and edit only that record.

● dBASE II

- The most annoying factors of dBase II are the arbitrary limitations on capacities of the system:

- 32 fields
- 1000 bytes/record
- Two files open at a time
- 64 memory variables available
- No capacity to do arrays
- Very limited mathematical capability.

- dBase II was originally written for an 8-bit processor.

The current version of dBase II is merely a ported version of the 8-bit design. Therefore dBase II does not take advantage of the 16-bit processor in the IBM PC.

Testing of the three packages shows that they are very similar in their ability to perform database query activities. The results are surprising in just how similar the three packages really are. The analysis of variance shows that no one package can be statistically determined to be superior to another. The differences in the packages reside more in the proposed usage of the system than in the technical capabilities of the systems. None of these packages is really made for the strictly novice user. They all require a modicum of familiarity with file concepts and programming concepts. This is true even to use the system in an interactive mode. Obviously, the level of expertise required to perform programming activities is even higher, but the effort expended to learn one of the packages is well worth it. Choose one and dig in! □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

## ES/P: A SYNTAX-CHECKING PASCAL EDITOR

*reviewed by Jim Gore*

*ES/P*
*Bellesoft, Inc.*
*2122 Bellevue Way SE*
*Bellevue, WA 98004*
*price $249 (Pascal version)*

A programmer's text editor can be a great productivity tool. Many have features that make entering and editing source code an easy and efficient task. Bellsoft has taken the idea one step further with Entry System for Programs (ES/P), an editor that combines standard editing functions with a new concept: language-oriented editing. Their products include a version for Pascal and C, with versions for ADA, COBOL, and FORTRAN under development. This review looks only at the Pascal version of ES/P.

ES/P should appeal to both experienced and beginning Pascal programmers. The experienced programmer saves time and keystrokes with ES/P, and the beginner learns the language structure faster. The editor automatically formats code while it's being entered, and enforces proper Pascal syntax as well.

SCOPE OF THE PRODUCT

ES/P provides three methods of entering Pascal code: stream mode, abbreviation mode, and expand/view mode. All three modes are available simultaneously.

Stream mode allows you to enter text character by character just like any other editor, except that your code is automatically formatted. Only proper Pascal syntax is allowed. When you enter a (syntactically appropriate) reserved word such as BEGIN, IF, FUNCTION, etc., the editor recognizes it and inserts a program skeleton into your code, complete with placeholders for the required and optional fields. Just entering the reserved word PROGRAM at the start of an ES/P file yields the following skeleton:

```
PROGRAM     =id= ( INPUT, OUTPUT ) ;
            =declarations*=
BEGIN
            =statements*=
END.
```

Abbreviation mode allows you to insert program skeletons or replace placeholders with a single keystroke. This is a very powerful feature of a program-oriented editing. Each placeholder in a program skeleton has a list of valid words or structures. If you press the ALT key and a one-character shorthand for a language structure, the current placeholder is replaced with actual Pascal. In the program skeleton above, the =declarations*= field can be expanded into a PROCEDURE skeleton with ALT-P; a FUNCTION skeleton is expanded with ALT-F; a TYPES skeleton with ALT-T, etc. All skeletons are properly indented, creating a clear and consistent program format.

Abbreviations are context-sensitive, making them easy to use and remember. Pressing ALT-C at the =declarations*= field will give you a CONST structure; pressing ALT-C at the =statements*= field will give you a CASE structure; and pressing ALT-C at the =simple-type= placeholder will give you the CHAR type.

The editor makes it easy to build entire program skeletons quickly before you go back and work on the details of each module. This works well with my own style of top-down program development. I usually start with an overall problem to solve, and continue to break my main objective down into precisely specified subproblems, assuring myself that if each subproblem is solved correctly, my original problem will also be solved. This editor won't take the place of adequate program design and 'pseudo-coding,' but it could make it easier to methodically develop and test modules for larger programs.

When you save a Pascal file, ES/P tells you if any required placeholders are left unfilled. A function key moves you to the next unfilled required placeholder.

Many of the keyboard's cursor movement keys perform language-oriented functions as well as standard editor movements (beginning of page, line, range, file, etc.). The TAB key advances the cursor to the next placeholder where Pascal input is allowed. Other keys move the cursor to the beginning or end of a program structure. Program structures can also be marked, copied, or deleted with single keystrokes.

The program comes with a template that fits around the keyboard, describing the functions performed by the PC's special function and cursor movement keys. If that is not enough for you, the program has an on-line HELP key, giving you on-screen reference to the editor's various keys and their functions.

In ES/P's expand/view mode, positioning the cursor inside a placeholder field activates the VIEW NEXT key, letting you step through the list of options available for that field. When you see the option you need, press the EXPAND key and a Pascal statement, structure, or keyword is inserted into your text. In the program skeleton above, the =statements*= field can be EXPANDed into: if-then-else, case, repeat, while, begin-end, with, readln, writeln, read, write, return, cycle, break, goto, label, procedure-call, or assignment.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Beginning Pascal programmers will undoubtedly appreciate this on-line prompting method of program entry. No longer will the beginner have to learn proper Pascal syntax through the "compile and error" method.

OTHER FEATURES OF ES/P

ES/P lets you work on up to four files simultaneously through on-screen windows. You can terminate its language-orientation mode before reading a file in, and you can have some windows act like a normal text processor while others enforce all the rules of Pascal. This is very convenient, since all files read into a Pascal-oriented window are checked and formatted for proper syntax. But it can be time consuming for long files, and the preliminary version I used often refused to read in some source code files, repeatedly displaying an "Input is not proper for the field. Continue?" message, even when I responded with "y".

Single keystrokes are used to insert and delete blocks of text. An INSERT WINDOW key lets you move all or part of one window's contents into another.

ES/P does not let you redefine or customize your keyboard. This is a big drawback if you have already mastered another editor's commands. You can, however, build MACROs consisting of long series of commands or keystrokes that are stored and executed with a MACRO EXECUTE key. MACROs can be saved in a file and used in a future editing session.

Another feature missing in this editor is a print command. If you are in the middle of an editing session and you want to print out a portion of some file, you're out of luck. You'll have to save your files and exit to DOS to print it.

Another complaint I have is with the FIND string command: it distinguishes between upper and lower case when searching for a matching string. You have to type a string exactly as it appears in the text in order to FIND it. "End," "end," and "END" are treated as different strings.

ES/P uses a "crash-proof" file system with the help of a "working file" on disk that stores changes to your text file as you make them. If no key has been pressed for six seconds, the program saves any changes to the working file. Before this occurs, pressing the UNDO key will get rid of any changes made since the last automatic save to the working file.

When you press the SAVE WORK key, ES/P uses the work file to create a new disk version of your file. The original version is renamed with a ".bak" extension, and the old backup is deleted. Should you suddenly experience a power failure, nothing is lost. The file(s) you were editing are recovered the next time you access them with ES/P (that is, assuming you can break old habits and forego the silent pleasure of editing files on a RAM disk).

While I was in the middle of one ES/P editing session, the program (for no reason that I could see) simply died, and not even CTRL-ALT-DEL would bring my PC back to life. But when I rebooted and returned to ES/P, my unnamed file was all there, nothing was lost.

Documentation for ES/P is adequate but not exceptional. I wish the manual included more examples, especially on language-oriented features and options. But the manual's tutorial lessons do a good job of introducing you to ES/P's basic features, and an appendix lists all of the Pascal skeletons and how/when they can be invoked using ALT key combinations.

What ES/P lacks in speed (watching ES/P read in and format a block of code from another file is like reading electronic messages at 300 Baud), it more than makes up for in its command of the Pascal language. It should eliminate syntax-related compiler errors completely, and its consistent program formatting will undoubtedly contribute to faster program debugging.

Working with an editor that enforces "correct" program entry takes a little getting used to. But for the serious Pascal programmer, ES/P is a valuable productivity tool.

```
(*   example of what 6 keystrokes can do in ES/P  *)

PROGRAM
  ~id~ ( INPUT, OUTPUT ) ;

  VAR
    ~var-defs~

  PROCEDURE ~id~ p(parameters)p ;
    pdeclarations*p
    BEGIN
      pstatements*p
    END ;

BEGIN

  WHILE ~condition~
  DO IF ~condition~
    THEN pstatement*p
    pelsep

END.

1 FILE+  <none>  PASCAL  LINE      9 OF     22
Can't expand existing range              CHAR   ENTER
```

□



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

# MISC.

## MACHINE PERCEPTION

If your interests or consulting work deal in hooking computers up in bizarre ways, write for a copy of SENSOR: the Journal of Machine Perception. It covers a wide range of sensory gadgets for machines, including computers. If you need some temperature and pressure reading instruments, magnetic proximity sensors, eddy current probes, or photon coupled interrupters, this is the place to find them. The journal is free to qualified applicants.

*SENSOR*
*North Am. Technology, Inc.*
*174 Concord St.*
*Peterborough, NH 03458*
*(603) 924-7136*

## ONLY THREE TYPES OF PROGRAMS

Yes, that's right, there are only three kinds of programs: the one the specifications called for, the one we actually managed to finish, and the one we dreamed about writing with all the bells and whistles.

## FRIENDS/ENEMIES/ANYBODY

Imagine for a moment that several subversive programmers joined together and offered totally unethical services for hire. They would be "mercenary programmers" in the true sense of the word. Consider the kind of search and destroy (or alter) missions that they could provide. Their purchase order sheet might look like this:

*Mercenary Programmers Anonymous (MPA's)*
*San Quentin Island*
*Fargo, ND 78787*

Send $10 for each service desired, circle the items requested. All numeric data you supply is assumed to be in decimal format unless otherwise indicated.

1. Erase your name and any additional information from all FBI/CIA files.
2. Change your S.S. Number.
3. Change someone else's S.S. Number.
4. Remove bad credit records. ($10 for each record.)
5. Insert bad credit records. ($10 for each record.)
6. Increase your S.S dividends.
7. Decrease someone's S.S. dividends.
8. Transfer funds into your Checking/Savings account.
9. Remove funds from someone's Checking/Savings account.
10. Tie up a corporation's phone system for 24 hours.
11. Generally mess up a company's on-line computer system.
12. Alter court documents.

## FASTER PASCAL

Software Building Blocks has released their very fast Pascal compiler for a price of $95. This includes their extensive and full featured version of standard Pascal. In comparison, SBB Pascal consistently surpasses Turbo Pascal in both source and EXE code size as well as in most benchmark tests. Certain standard Pascal features and many desirable extensions to the language are found in SBB's version but are absent from Turbo's. Once you have written your Pascal programs, you can upgrade to SBB's full development package for $350 and get the advantages of separate compilation, overlays, BCD arithmetic, 8087 support, and run-time source. For complete comparison literature, contact:

*Software Building Blocks*
*P.O. Box 119*
*Ithaca, NY 14851*
*(607) 272-2807*

## FAMOUS PHRASES

If you are working on a computer system and are having trouble explaining to the client the reason for the extra charges, these two phrases may come in handy: "Well, we got a little bogged down because your buffer bearings needed greasing." Or how about this one: "Usually it doesn't take quite this long, but your I/O jets were badly clogged and they had to be resurfaced."

## CULT WARE

Attention microcomputer owners! You can now start your own Cult. A new software package called CultStar is guaranteed to bring in bucks and bozos. CultStar is a menu-driven text-writing program that makes you into a prolific cult figure OVERNIGHT! Just type in answers to simple questions and watch your computer churn out endless pages of customized explanations, autobiographies, predictions, operation manuals for regional centers and letters you have received from the masses. We can also help you get going with our subscription MassMail service. Order Today! CultStar ($599). MassMail ($599).

RipCult, our database option, gives CultStar the ability to borrow all of the ideas and buzzwords of other successful cults and disguise them for your own use. Includes a 200 page manual. RipCult ($499).

CultAcct is our accounting package that sets membership prices (what the market will bear, which will floor you), and does all needed bookkeeping. Includes PadAccount (tm) our Expense Account diddler. Comes complete with a 130 page manual explaining how to become tax exempt, how to "die" and leave control of the Church (or whatever) with a fictitious foundation, etc. CultAcct ($299). □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

# ARE YOU AN ISSUE SHY OF A FULL SET?

Complete your library! Order any back issues you may have missed — or become a new subscriber. Use the form below or the handy subscription envelope provided. You'll be glad you did.

- Vol. 1, No. 1: Norton Pascal I/O, Program Style Manual, COBOL, System Analysis, BASIC modules, Rounding error.
- Vol. 1, No. 2: Defining character sets, C86 BIOS calls, Copy protection, Pascal code, Input routines, Writing an Assembler.
- Vol. 1, No. 3: ASM dumb terminal, C source code, ASM graphics line drawing, Text editors reviewed, Utilities.
- Vol. 1, No. 4: Diskette directory via Pascal, MASM WHILE-WEND, C/ASM interface, ASM stay resident routine, Structured programming.
- Vol. 1, No. 5: Multi-tasking software review, Transporting C Programs, Programmers calculator, dBase bargraphs, Utilities.
- Vol. 1, No. 6: ASM printer configuration, DOS patch utility, Hi-speed graphics, Software ownership, dWINDOW review.
- Vol. 2, No. 1: Freelance consulting, Sorts in C, Pascal text editor source code, Designing national database, Ultra utilities.

---

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

Telephone _____

Send me the following back issues:
($4 each U.S., $5 Canada, $7 Other, U.S. funds)

_____ Vol. 1, No. 1
_____ Vol. 1, No. 2
_____ Vol. 1, No. 3
_____ Vol. 1, No. 4
_____ Vol. 1, No. 5
_____ Vol. 1, No. 6
_____ Vol. 2, No. 1
_____ Vol. 2, No. 2
_____ Vol. 2, No. 3

_____ 1 year bi-monthly subscription
at $24.00 ($30 Canada, $40 other, U.S. funds only)

_____ Check enclosed _____ Bill me

_____ Charge my Bankcard: ___ VISA ___ MC ___ Am Ex

# _____ Expir. Date _____

Signature _____

Send to:
PROGRAMMER'S JOURNAL
P.O. Box 30160
Eugene, OR 97403
(503) 345-3043

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

*Busted Basic*

*P Mate Macros*

*C Utility Routines*

*Syntax Checking Editor*

*8088 Assembly Macros*

*dBase II / KMAN / R:base*

*The Future of Disk Protection*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED